ARMY RESEARCH LABORATORY

Modifying ModSAF Terrain Databases to Support the Evaluation of Small Weapons Platforms in Tactical Scenarios

MaryAnne Fields

ARL-TR-1996 AUGUST 1999

Approved for public release; distribution is unlimited.

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5066

ARL-TR-1996 August 1999

Modifying ModSAF Terrain Databases to Support the Evaluation of Small Weapons Platforms in Tactical Scenarios

MaryAnne Fields
Weapons & Materials Research Directorate

Abstract

In this report, we describe tools for the creation and modification of modular semi-autonomous forces (ModSAF) terrain databases to support the evaluation of a small autonomous robot in a tactical scenario. Our work is motivated by the modeling and simulation needs of the Demo III robotics program which is developing a small tactical robot called the experimental unmanned vehicle (XUV). The XUV is a small wheeled robot which must autonomously navigate through its environment. The primary mission of the XUV will be to augment the scout forces, so it must provide reconnaissance, surveillance, and target acquisition information (RSTA) to its operators. Modeling the XUV in a simulated environment is challenging since existing terrain databases do not have sufficient resolution to examine the mobility characteristics of small vehicles.

Our tools increase the resolution and detail of existing terrain databases so that the databases have sufficient detail to challenge the mobility, chassis dynamics, and RSTA models of a small unmanned platform. To properly model a small vehicle such as the XUV, the terrain database in ModSAF needs to be modified. The modification is done in two phases. In the first phase, the resolution of the grid underlying the terrain is increased by placing additional elevation grid posts between the existing posts. Elevations are assigned to the new grid posts using mathematical terrain models such as the variable resolution terrain Model (Wald & Patterson, 1992). The new, higher resolution terrain directly affects the vehicle dynamics and the line-of-sight algorithms. The new terrain does not directly affect the ModSAF route-planning algorithms. In the second phase of our terrain database modifications, the slopes on the new terrain are examined. Regions that are steep or inaccessible to the XUV are marked as obstacles in the database. The route-planning algorithms use these "obstacles" to avoid planning routes through regions that are too steep for the XUV.

TABLE OF CONTENTS

INTENTIONALLY LEFT BLANK

LIST OF FIGURES

INTENTIONALLY LEFT BLANK

LIST OF TABLES

INTENTIONALLY LEFT BLANK

MODIFYING MODSAF TERRAIN DATABASES TO SUPPORT THE EVALUATION OF SMALL WEAPONS PLATFORMS IN TACTICAL SCENARIOS

1. INTRODUCTION

In this report, we describe tools for the creation and modification of modular semi-autonomous forces (ModSAF) terrain databases to support the evaluation of a small autonomous robot in a tactical scenario. Our work is motivated by the modeling and simulation needs of the Demo III robotics program. Under the Demo III robotics program, the U.S. Army is developing a small, survivable, experimental unmanned ground vehicle (XUV) (see Figure 1) capable of autonomous operation over rugged terrain as a part of a mixed military force containing both manned and unmanned vehicles. The primary role of the XUV will be to augment the Army battalion and brigade task forces scout platoon. In this scout mission, the XUV is expected to move through the terrain using proper military movement techniques and with minimal human oversight. Using its reconnaissance, surveillance, and target acquisition (RSTA) package, the XUV can acquire information about the disposition of enemy forces. We must include "models" of XUV chassis dynamics, XUV mobility, and the XUV RSTA package to properly assess the contribution of the robot to the overall mission. It is also important to use terrain databases that have sufficient detail to challenge the mobility, chassis dynamics, and RSTA models.

Figure 1. The XUV.

To properly model a small vehicle such as the XUV, the terrain database in ModSAF needs to be modified. The modification is done in two phases. In the first phase, the resolution of the grid underlying the terrain is increased by placing additional elevation grid posts between the existing posts. Elevations are assigned to the new grid posts using mathematical terrain models such as the variable resolution terrain model (Wald & Patterson 1992). The new, higher resolution, terrain directly affects the vehicle dynamics and the line-of-sight (LOS) algorithms. The new terrain does not directly affect the ModSAF route-planning algorithms. In the second phase of our terrain database modifications, the slopes on the new terrain are examined. Regions that are steep or inaccessible to the XUV are marked as obstacles in the database. The route-planning algorithms use these "obstacles" to avoid planning routes through regions that are too steep for the XUV.

Many ModSAF terrain databases, which the developers refer to as compact terrain databases (CTDBs), have elevation posts spaced uniformly 125 meters apart. At this resolution, it is difficult to examine mobility issues. As an example, consider the effect of terrain slope on cross-country travel at Fort Hood, Texas. Figure 2 gives the change in elevation from one elevation post to the next required to produce the desired slope across the elevation grid square. The change in elevation required to produce a 30° slope on a 125-meter grid square (72.2 m) is more likely to occur in mountainous regions where wheeled vehicles are not likely to go. The 125-meter resolution database representing Fort Hood is almost flat in most spots, presenting no significant slopes to climb. However, Figure 3 illustrates that even within a single grid square, there may be significant slopes to climb such as those associated with eroded areas, ditches, or culverts. With the length of the vehicle being used to estimate the dimensions of the terrain area shown in the picture, the photograph shows a 100- by 100-meter area of the terrain. This area would be represented in most terrain databases as a single flat square.

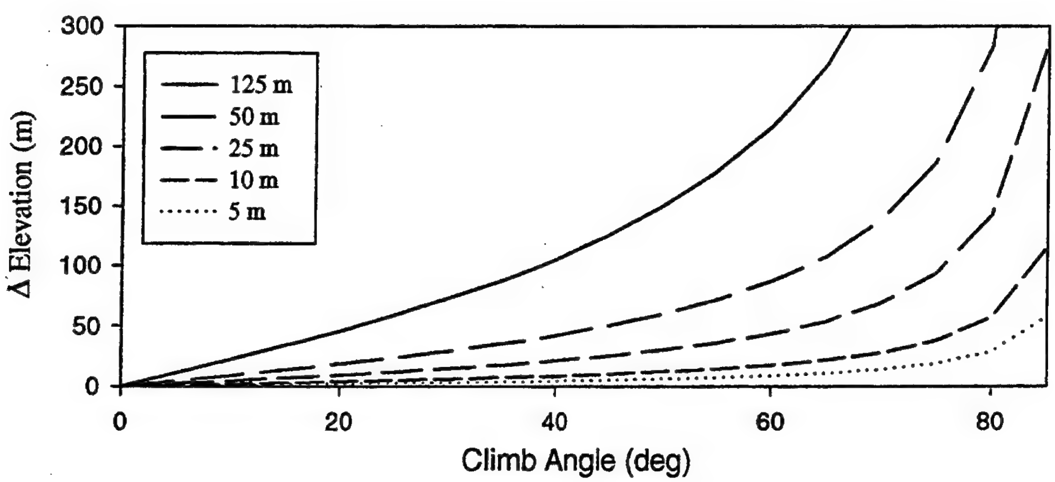


Figure 2. Change in Elevation as a Function of Climb Angle.

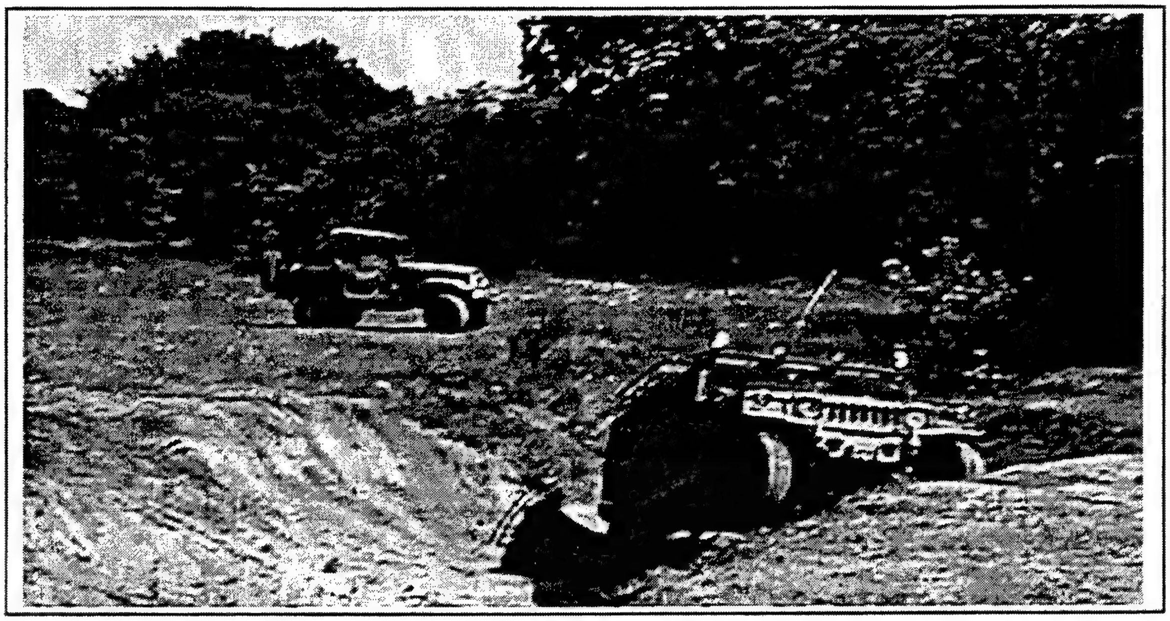


Figure 3. A 100-meter Patch of Terrain at Fort Hood, Texas.

There is information in a CTDB database concerning trafficability. Each grid square has a soil type associated with it. By using a program such as the North Atlantic Treaty Organization (NATO) reference mobility model (NRMM), (Ahlvin 1992), users can adjust the maximum speed of their vehicle for each soil type. In addition, there is an abstract layer in CTDB databases that consists of polygons representing buildings, lakes, tree canopies, impassable regions, and other features on top of the actual terrain skin. These polygons interact with the route-planning algorithms by providing potential obstacles. They also can interact with the visibility algorithms by blocking the LOS between points. As far as mobility is concerned, there are only two possible responses to each type of abstract region: a class of vehicle may avoid all polygons of a given type or may ignore all polygons of that type.

This behavior is a little simplistic for vehicles that travel through, instead of around, the tree canopies. Canopy polygons do not affect the speed or the route of the vehicles that travel through them. In addition, all canopy polygons on a given terrain database have the same effect on vehicle mobility. An important parameter of the canopy that is not considered for mobility evaluation is tree density (density is considered in the visibility analysis). Cultivated tree farms and orchards, with uniformly spaced trees and little underbrush, should be easier to move through than natural wooded areas with variably spaced trees and lots of underbrush.

In evaluating the potential contribution of the XUV to the scout mission, it is important that we consider the impact of the XUV's mobility on the mission. This is done by first

increasing the resolution of the terrain database. The information in the high-resolution terrain database is used to adjust the soil types and to create abstract regions to decelerate and redirect the XUV. This report presents only the methods used to modify the ModSAF terrain databases to make them more useful for the evaluation of small vehicles. Actual evaluations of the XUV will be presented in future reports.

2. INCREASING THE RESOLUTION OF THE ELEVATION GRID

Increasing the resolution of the elevation grid is, in theory, a relatively easy process. We simply need to find a higher resolution terrain database for our battlefield and use it to construct the new elevation grid. However, it is not always possible to find such a database. Standard National Imagery and Mapping Agency digital terrain elevation data (DETED) databases covering most of the world use 100-meter grid posts. As a substitute for measured elevation data, we can artificially increase the resolution of the database by fitting the original elevation posts with a mathematical function that fits the measured elevation posts to the desired accuracy and gives statistically realistic surface variations between the measured posts. One such function is variable resolution terrain (VRT) model.

The VRT model is a continuous, differentiable surface generated by summing several simpler surfaces referred to as hills. The equation of a hill function defined on the two-dimensional space of real numbers, $\Re^2$, is written as

$$h(x,y) = \alpha e^{f(\|(x,y),(\xi,\eta)\|_d)}. \qquad (1)$$

Here, $\alpha \in \Re, (\xi,\eta) \in \Re^2$ and $\|(x,y),(\xi,\eta)\|_d$ is a metric on $\Re^2$. The most familiar metric is the Euclidian distance between points. The Euclidian metric gives relatively smooth hills, so other metrics are often used to produce hills with sharper peaks.

By varying the metric, the function f, and the parameter α, it is possible to generate hills of any size and shape. To create a generic VRT surface, hills of various sizes and shapes are combined using the principle of self-similarity. Self-similar objects are invariant with respect to scale so that a portion of the object, viewed at the proper magnification resembles the whole object. In a generic VRT surface, the distribution of hills is statistically self-similar. More details of the VRT model are given in the paper by Wald and Patterson (1992) and the later papers by Wald (1994, 1995). These later papers discuss methods used to fit existing terrain databases with VRT surfaces. The same software tools we describe in this report can also create new

terrain databases for ModSAF with specific characteristics designed to test movement algorithms developed to model the movement of small vehicles within a ModSAF exercise.

The software we have developed is based on software and documentation available in the ModSAF developers' kit (Braudaway et al. 1996). CTDBs use compression methods to store terrain databases in much less space than other terrain database formats require. In gridded databases, the information about the terrain surface is stored independently from detailed information about the feature layer on top of the surface. For each elevation post, the elevation, two soil types, and flags indicating the presence or absence of features such as buildings, trees, roads, or canopies are stored in 32-bit words. The elevation grid is broken into patches (a square region typically four posts long by four posts wide). Each patch contains detailed feature information, such as the location and size of buildings or the location and width of roads. Features that pass through more than one patch must be subdivided so that a portion of the feature is stored on each patch.

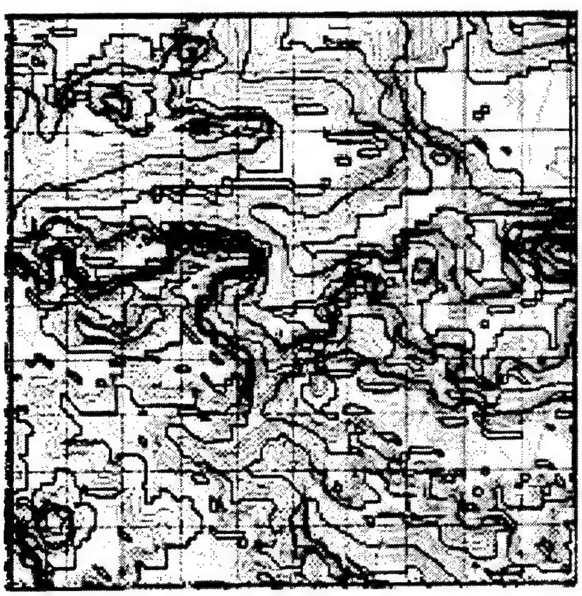
Much of our program ***add_vrt.c*** (given in Appendix A) is based on the program ***recompile.c*** in the CTDB library in the ModSAF developers' kit. ***Recompile.c*** converts older CTDB databases to the current format. It can also add features such as additional roads, bridges, or canopies to CTDB databases. We have extended this code so that we can change the resolution of the CTDB databases. With our new program, we can change the soil types of the elevation posts to affect the movement of the vehicles on that post, simulating rough or rocky regions on the terrain. Impassable regions can be added to the databases, especially inside the existing canopies, thus forcing vehicles to modify their plans to avoid the impassable regions.

Increasing the resolution of a database is relatively easy. The C routines ***get_original_elevations***, ***look_for_features***, and ***look_for_canopy_and_lakes***, store information from the original database so that it can be used to construct the new database. A major difference between our code and ***recompile.c*** is that we must recompute the feature flags for each post in the new database. For example, suppose a road intersects a post on the original 125-meter database. At 25 meters' resolution, this post is covered by a 5 by 5 square of 25-meter posts. It is unlikely that the road will intersect all 25 of these smaller posts. These flags are used to streamline search routines for the LOS and planning algorithms; it is not advisable to simply pass the flags from the old posts to the new ones.

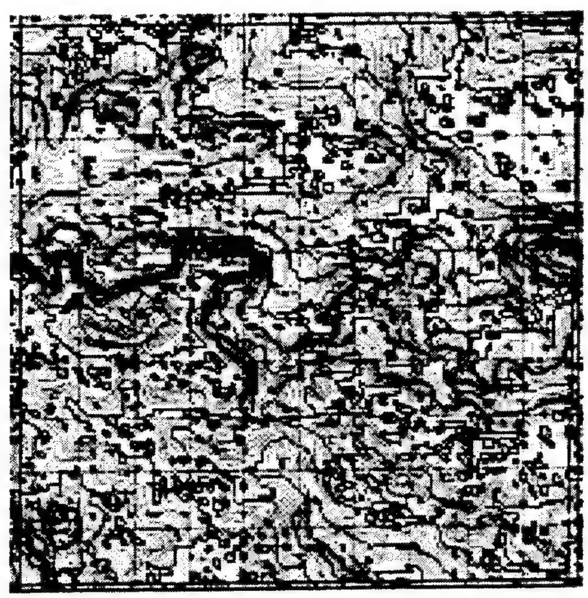
Our fitting routine is contained in the C routine ***make_vrt_hills***. This routine uses the fitting method outlined by Wald (1994). In his method, the VRT surface is determined

iteratively. The highest elevation post on the database determines the height and location of the first hill. The parameters for the first hill are chosen to minimize the differences between the elevation grid and this hill. Next, we compute the differences between the elevation posts and the first hill. The largest of these differences determines the height and location of the second hill. The process continues until the difference between the each of the elevation posts and the height of the VRT surface at that point is small enough.

To produce a new terrain database, we must write the new elevation grid and rewrite all the feature information. The routine ***write_elevation_posts*** transfers the refined elevation grid to the new database. The C functions, ***collect_trees***, ***collect_canopies***, ***collect_buildings***, and ***collect_linears***, allow us to collect the features from the original database. These routines were modified from their original form in ***recompile.c.*** so that features could be properly subdivided for the new patch size. Figure 4A shows a contour map of a 10-kilometer by 10-kilometer section of Fort Hood, sampled at a 125-meter resolution. The contours are equally spaced, 2 meters apart. Figure 4B shows the same section enhanced with VRT hills, sampled at 50 meters' resolution. The enhanced database has a number of small hills and valleys that are not present in the original database. The heights of the original elevation posts have been preserved.



A



B

Figure 4. A 10-kilometer by 10-kilometer Section of the Fort Hood Database (2-meter contours). (A shows the original 125-meter database; B shows the VRT-enhanced 50-meter database.)

Using this program, it is possible to create very high-resolution databases. However, there is a trade-off between database resolution and usability. Tests run at Fort Knox (Nida, 1998) indicate that it is not practical to use extremely high resolution databases in large ModSAF exercises involving hundreds of simulated entities. The amount of time ModSAF spends processing terrain information increases to the point that operators must wait a long time for tactical display screens to be redrawn. Also, the linear networks that describe the roads, rivers, and other linear features become unwieldy since there must be at least two points on each patch that intersects them. The planning algorithms limit the number of path points that can be stored for a given plan. This number can be increased but not to the point that it can do much good.

At this time, there are four solutions to this problem. First, we can limit the dimensions of the battlefield. The Fort Knox tests involved databases that represented a 50-kilometer by 50-kilometer section of Fort Hood, Texas. In its original 125-meter resolution, the database was represented by a 400 by 400 elevation grid and 10,000 500-meter patches containing feature data. At 50 meters' resolution, the elevation grid was a 1000 by 1000 matrix and the feature data were contained in 62,500 200-meter patches. By reducing the dimensions of the battlefield, the amount of data becomes easier to handle. This solution is well suited for studies involving small vignettes in which the entities are confined to a small portion of the battlefield.

The second possible improvement is to increase the dimensions of the patch. This option was discarded since it affects the portability of databases that we create. A potential user would be forced to reset a parameter in the ModSAF code to use the databases. Also, the dimensions of a patch were chosen by the ModSAF developers to optimize memory usage. Changing the dimensions may decelerate rather than accelerate the LOS and route-planning algorithms.

The third option is to use ModSAF micro-terrain to add more detail to the terrain surface. Micro-terrain is a layer of triangular facets on top of the terrain surface. It is used to describe small terrain features, such as single boulders or berms. It is also used to describe multilevel surfaces such as underpasses and overhangs. There are limits on the micro-terrain approach. First, any changes should stay within a single patch so the added hills must effectively vanish outside this patch. There is also a limit on the number of triangles that can be added to a database. The fourth option is to create databases based on triangularized irregular networks (TINs) rather than uniformly spaced grids. The triangles that comprise the TIN are not uniform in size. Large triangles are used in relatively flat, featureless regions of the terrain. Small triangles are used in rough terrain. The non-uniform size allows posts to be concentrated where greater detail is required. The TIN representation of terrain is actually an extension of the method used

to add micro-terrain. In TIN databases, there is no elevation grid and the entire surface is represented by triangular facets. We are still examining this option.

3. ADDING ABSTRACT REGIONS AND ADJUSTING SOIL TYPES

Refining the elevation grid in a CTDB database increases the probability that a ModSAF vehicle will encounter a significantly steep region of the terrain as it travels cross country. Unfortunately, the elevation grid is not used by the route-planning algorithms so the vehicle cannot avoid a steep spot that is represented only by the elevation grid. In ModSAF, the planning routines use the abstract layer of the terrain database to plan the route. This layer consists of polygons representing features such as tree canopies and lakes, and line segments representing features such as railroads, power lines, and fences. By using the simplified terrain representation in the abstract layer, the route-planning algorithms efficiently plan routes that avoid these regions.

We have to add polygons to the abstract layer of the CTDB database to force the vehicle to "see" the steep regions. The routine ***add_steep_regions***, shown in Appendix A, is used to add steep regions to the terrain database. This routine uses the dimensions of the VRT hills to assign a "steepness" value to the entire hill. Steep VRT hills are enclosed in abstract soil regions. These abstract soil regions are simply polygons with an additional parameter that specifies a soil type for the entire region. Typically, these regions are used to designate lakes in the database. By specifying another soil type, we can use these abstract soil regions to control the movement of vehicles on dry land. By altering the vehicle parameter file, it is possible to force specific types of vehicles to avoid these regions. An example of a vehicle parameter file is shown in Appendix B. In the terrain analysis section of the entity parameter file, the avoidance mask includes the TERRAIN_WATER flag which directs the vehicle to avoid soil regions. The specific types of soil regions are given in the *avoid_soils* subsection. In this section, the soil type is referenced by a binary code. This 32-bit integer contains mobility parameters and the Simnet and close combat tactical trainer (CCTT) indices for the soil type. Table 1 shows a map of the bits for the binary soil code.

Table 1. Bit Definition for the Binary Soil Code

Bits 31-13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Road (R)	Slow Go (S)	No Go (N)	Water (W)									
Unused	Mobility parameters				CCTT soil index					Simnet soil index			

Table 2 gives the indices, description, and binary code for the 16 most common soil types. Generally, CTDB databases use the Simnet soil types, although it is possible to define additional soils.

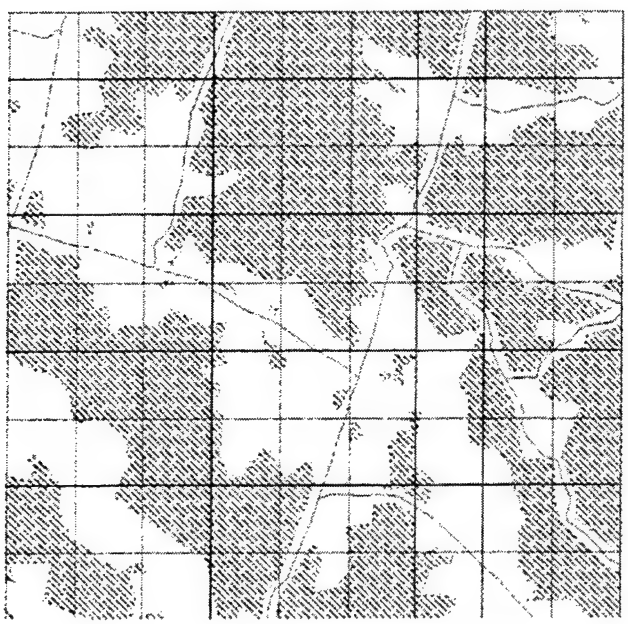
Table 2. ModSAF's Soil Types

Simnet index	CCTT index	Soil type description	Mobility parameters				Binary code
			R	S	N	W	
0	0	Undefined					0
1	20	Pavement, concrete, asphalt	*				4417
2	17	Dry loose surface road	*				4370
3	5	Soft					83
4	28	Water 60-inch depth			*	*	1988
5	26	Water 16-inch depth		*		*	2981
6	4	Very soft		*			2118
7	18	Wet loose surface road	*	*			6439
8	15	Very hard (slippery)		*			2296
9	19	Swamps, bogs—slow go		*			2361
10	7	Hard					2170
11	7	Hard					123
12	7	Hard					124
13	22	Brush land		*			2413
14	9	Very hard			*		1182
15	19	Swamps bogs—no go			*		1343

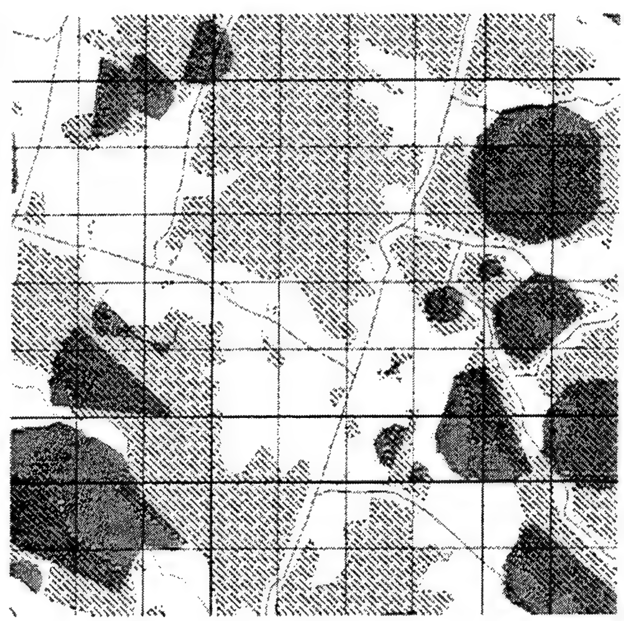
To investigate the performance of the XUV in a scout mission, it may be necessary to allow the vehicles to move through the tree canopies. Tree canopies are polygonal regions in the abstract layer; they can be treated like obstacles by a class of vehicles or ignored. Treating the canopies as obstacles is realistic for large vehicles. Ignoring the canopies so that vehicles can move through the areas of the battlefield covered by canopies is not realistic. It assumes that vehicles can negotiate any region of the canopy at the same rate of speed. The routine ***look_for_canopy_and_lakes*** can be used to adjust the soil parameters of the underlying posts within the canopies. By changing the

mobility parameters in the vehicle parameter file, vehicles will decelerate inside the canopies. However, the vehicles will still drive in straight lines. In the mobility parameters section of the vehicle parameter file given in Appendix B, the vehicle decelerates on water (soils 4 and 5) and on soils 9 through 15. In this section, the soils are referenced by their Simnet soil index which is shown in Table 2. By adding obstacles to the canopies, the vehicles will use realistic routes through the canopies. The routine ***add_subcanopies*** alters the canopies by adding artificially generated impassable regions to the canopies. Just like the steep regions, the impassable regions are abstract soil regions—only the soil parameter is different. An example of an altered canopy regions is shown in Figure 5. The original canopies are represented by the light gray hatched polygons in Figure 5a and 5b. The sub-canopy regions shown as dark gray polygons in Figure 5b. In this particular example, the sub-canopies are relatively large but they do not need to be. CTDB databases do not have limits on the size of the polygons; however, there are limits on the total number of vertices.

As an alternative to the sub-canopies, we can replace the canopies with distribution of trees and tree lines. This approach is ideal for small battlefields since it affects both the LOS and the mobility algorithms. On large high-resolution battlefields, it adds a significant number of features to the patch data.



A



B

Figure 5. Adding Subcanopies to a CTDB Database. (Figure 5a shows the original ModSAF canopies; Figure 5b shows subcanopies in dark gray.)

4. CONCLUSIONS

We have designed tools to modify CTDB terrain databases to support the evaluation of a small autonomous robot in a tactical scenario. The first tool increases the resolution of the elevation grid. This modification has the most effect on the vehicle dynamics model and the LOS algorithms. The second set of tools adds polygonal regions to the abstract layer of the terrain database. These modifications affect the planning algorithms.

We have begun to use these databases in our examination of the XUV in tactical scenarios. Tests were conducted at Fort Knox during the summer of 1998 to determine the feasibility of using high resolution databases in large ModSAF exercises. Although the results were somewhat disappointing, they have given us some useful information about the use of high resolution databases. By using ModSAF micro-terrain or a TIN database, we may be able to increase the resolution of the elevation grid substantially without the problems of the gridded approach.

By using steep regions and the sub-canopy regions, we can generate more realistic paths for the vehicles. However, these regions are based on artificial modifications of the terrain database. It is important that we consider several statistical variations of the same battlefield in a study to avoid results that hinge on the exact location or shape of these regions.

INTENTIONALLY LEFT BLANK

5. REFERENCES

Ahlvin, R.B., and P.W. Haley, "NATO Reference Mobility Model," GL-92-19, U.S. Army Waterways Experiment Station, Vicksburg, MI, December 1992.

Braudaway, W., C.G. Buettner, F.L. Chamberlain, A. Evans, M. Longtin, J.E. Smith, and T. Stanzione, "LibCTDB, Compact Terrain Database User Manual and Report," Naval Research Laboratory, Contract Number N00014-92-C-2150, 1996.

Nida Jon, personal communication, July 1998.

Sagacitech, LLC, "ModSAF3.0 Developer's Course Workbook," Monterey, CA, 1997.

Smith, J., "LibEntity Programmer's Reference Manual," Naval Research Laboratory, Contract Number N00014-92-C-2150, 1996.

Wald, J.K., and C.J. Patterson, "A Variable Resolution Terrain Model for Combat Simulation," BRL-TR-3374, U.S. Army Ballistic Research Laboratory, Aberdeen Proving Ground, MD, July 1992.

Wald, J.K., "Solving the 'Inverse' Problem in Terrain Modeling," ARL-TR-605, U.S. Army Research Laboratory, Aberdeen Proving Ground, MD, October 1994.

Wald, J.K., "Modeling Micro Terrain using the Variable Resolution Terrain Model," ARL-TR-866, U.S. Army Research Laboratory, Aberdeen Proving Ground, MD, September 1995.

INTENTIONALLY LEFT BLANK

APPENDIX A

THE ADD_VRT.C CODE

INTENTIONALLY LEFT BLANK

THE ADD_VRT.C CODE

A.1 Globals

Globals

```
#ifndef lint
    static char rcsid [] = "$RCSfile$ $Revision$ $State$";
#endif
#define DEBUG
#define READ_MES
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <libcmdline.h>
#include <math.h>
#ifndef sgi
    # ifndef MAP_AUTOGROW
    # define MAP_AUTOGROW 0
# endif
#endif
#define TMPFILE "recompile.tmp"
#define PHYSICAL_FEATURE_SIZE 40*1024*1024
#define MAX_NUM_LINEAR_MODELS 400000
#define MAX_NUM_AGGREGATE_MODELS 50000
#define DEFAULT_TRUNK_RADIUS 0.5 /* meters */
#define DEFAULT_FULLNESS 0.95
#define NO_FEATURES -1
#include <stdio.h>
#include <errno.h>
#include <stdstring.h> /*common/include/global*/
#include <libctdb.h>
#include <ct_feat.h>
#include "ct_cmplr.h"
#include "ct_reform.h"
#include <ct_post.h>
#include <libreader.h>
#include <stdalloc.h>
#include <stdext.h>
#include <time.h>
#include <string.h>
#include <libcoordinates.h>
#include <libgcs.h>
#include <curses.h>
#define my_factor 1
int32 my_patch_x,my_patch_y;
struct MY_HDR { float64 incrD,inv_incr,patch_incrD
float64 max_z,min_z, patch_inv_incr,post_increment;
int32 incr,patch_incr,max_x,min_x,max_y,min_y,max_x_post,
int32 min_x_post, min_y_post,max_y_post;
int32 pages_wide,pages_high,patches_wide,patches_high;} my_hdr;
#define INPUT_MEMORY 0
#define START_CELLS 50
static CTDB_CMP *ctdb_in, ctdb_out;
```

The boldface type indicates include files that can be found in the ModSAF3.0 source code.

```
static int32 output_format;
static int32 input_format;
static int32 feature_offset = 0;
static int32 mf;static CTDB_COMPILER_QUAD quad_root;
static int32 num_corrections = 0;
static READER_UNION *corrections;
static int32 output_db_patches_wide = 0;
static int32 output_db_patches_high = 0;
static int32 output_db_incr = 0;
static int32 output_db_patch_incr = 0;
static int32 offset_x = 0, offset_y = 0;
static int32 offset_x_meters = 0, offset_y_meters = 0;
static int32 min_patch_x, max_patch_x, min_patch_y, max_patch_y;
static int32 recompile_buildings = 0;
static int32 recompile_linears = 0;
static int32 recompile_trees = 0;
static float64 indb_south_lat, indb_north_lat, indb_west_long, indb_east_long;
static int32 gcs_mode;
static COORD_TCC_PTR tcc;
static int32 cur_cell;
static int32 water_fix;
static CTDB_DBASE_TYPE dbase_type = CTDB_HYBRID;
static int32 te_offset=0;
static int32 total_tes=0;
static int32 trans_end = 0;
static FILE *fptr, *color_file;
static TRANSITION_PATCH_INFO *trans_patches = NULL;
static void look_for_features();
static void add_canopies();
static int32 encode_physical_features();
static void collect_nodes_and_edges();
static void encode_abstract_features();
static void complete_micro_poly();
static void collect_microterrain();
static void collect_buildings();
static void collect_trees();
static void collect_canopies();
static void collect_linear();
static int32 *transform_posts();
static void fit_vrt();
static double vrt();
static double single_hill();
#define CMP_MAX_REF 1000
#define CTDB_INVALID_REF -1
typedef struct cmp_ref
    {
        int32 user_id;
        int32 index;
    } CMP_REF;
struct { CMD_STRING_OPTION input_ctdb;
 CMD_STRING_OPTION output_base_name; } options =
    {
     {"Input Ctdb", "Absolute pathname to the input CTDB" ,
     NULL , CMD_STRING , "input_ctdb" , NULL , "knox-0311.ctb" , },
     {"Output Name Base", "Output database base name" , NULL ,CMD_STRING , "output_base_name" ,
     NULL , NULL , }
```

In this section of code, the bold print specifies a function prototype. The code for these functions is found later in this appendix.

```
};

struct PARAMETERS {double cx,cy,hgt,angle,wgtx,wgty,power,w1,w2;}
    hill[1000],patch_hills[100][100][10];
int nhills, saved_pages_high,saved_pages_wide;
float64 hgt_max,original[500][500] = { 0.0};
float64 temp[500][500], mat_step = 10.0;
short int my_soil[500][500] = {0,canopy[5500][5500] = {0};
short int hills_per_patch[100][100];
```

A.2 Main

```
int main ( int argc, argv_t argv)
  {
   int32 result;
   CTDB_FILE_HEADER_CMP hdr;
   char **correction_files, first_db_name[CTDB_NAME_LENGTH], *input_db_name;
   int32 num_correction_files, header_assigned = FALSE,i;
   int32 *elev_ptr = ctdb_out.elevations,*soil_ptr = ctdb_out.soils;
   extern int atoi();
   char dbname,s;
   int32 leftover_argc;
   argv_t leftover_argv;
   char *rest_of_str, *cpy_rest_of_str, *current_file;
   int32 err, chars_processed;
   char sp = '\040';
   int32 *cells = NULL,ncells,cell_idx;
   float64 cell_south_lat, cell_north_lat, cell_west_long, cell_east_long;
   float64 cell_min_gcs_x, cell_max_gcs_x, cell_min_gcs_y, cell_max_gcs_y;
   float64 cell_min_utm_x, cell_max_utm_x, cell_min_utm_y, cell_max_utm_y;
   float64 cell_origin_x, cell_origin_y;
   int32 *patch_water_state = NULL;
   int ix,iy tloc;
   float64 loc_x,loc_y;
   uint32 p;
   int32 norm_x, norm_y;
   bzero((void*)&quad_root, sizeof(quad_root));
   cmd_process_options(argc, argv, &leftover_argc, &leftover_argv,
     (CMD_OPTION *)&options, sizeof(options), TRUE);
   cmd_gripe(leftover_argc, leftover_argv);
   gcs_mode = CTDB_MODE_SIMNET;
   reader_init(0);
   coord_set_tcc_awareness(TRUE);
   gcs_init(coord_get_datum_info);
   gcs_set_cell_awareness(GCS_MULTI_CELL);
   ctdb_cmplr_init();
   num_correction_files = 0;
   ctdb_in = ctdb_cmplr_read(options.input_ctdb.value, &input_format);
   ctdb_set_tiling_mode(FALSE);
   ctdb_intern_print_description(ctdb_in);
   ctdb_in->max_z = -999999.0;
   ctdb_in->min_z = 999999.0;
   look_for_features(patch_water_state);
   for (iy=0; iy<=ctdb_in->max_y_post; iy++)
     {
```

This section processes the command line argument and initializes coordinate system routines.

This loop reads the elevations from the original terrain database, specified by ctdb_in. These elevations are stored in the array original. The soil for each grid post is initially set at a constant value. The soil is modified by transform_posts.

```
        loc_y = iy*ctdb_in->incrD;
        for (ix=0; ix<=ctdb_in->max_x_post; ix++)
          {
            loc_x = ix*ctdb_in->incrD;
            original[ix][iy] = ctdb_intern_get_ground_elevation
                (ctdb_in,loc_x,loc_y,FAVOR_TES);
            if ( original[ix][iy] > ctdb_in->max_z)
                ctdb_in->max_z = original[ix][iy];
            if ( original[ix][iy] < ctdb_in->min_z)
                ctdb_in->min_z = original[ix][iy];
            my_soil[ix][iy] = 131074;
          }
    }

  hgt_max = ctdb_in->max_z - ctdb_in->min_z;
  output_format = CTDB_FILE_FORMAT;
  dbase_type = CTDB_GRIDDED;
  tcc = coord_define_tcc(COORD_UTM_NE,
        ctdb_in->origin_northing,ctdb_in->origin_easting,
        ctdb_in->origin_zone_number,ctdb_in->origin_zone_letter,
        ctdb_in->datum,(ctdb_in->max_x - ctdb_in->min_x),
        (ctdb_in->max_y - ctdb_in->min_y));
  get_ll_bounds(COORD_TCC, GCS_ILLEGAL_CELL, tcc,
        (float64)ctdb_in->min_x, (float64)ctdb_in->min_y,
        (float64)ctdb_in->max_x, (float64)ctdb_in->max_y,
        &indb_west_long, &indb_south_lat,
        &indb_east_long, &indb_north_lat);
  ncells = START_CELLS;
  if(!cells) cells = (int32 *)STDALLOC(ncells * sizeof(int32));
  if (!gcs_extent_to_cell_list(indb_south_lat, indb_west_long,
    indb_north_lat, indb_east_long,
    &ncells, cells))
    {
      cells = (int32 *)STDREALLOC(cells, ncells * sizeof(int32));
    }
  cells[0] = GCS_ILLEGAL_CELL;
  ncells = 1;
  cur_cell = cells[0];
  dbname = ctdb_generate_tdb_filename(CTDB_MODE_SIMNET,
    options.tdb_path.value,
    options.output_base_name.value,
    output_format, GCS_ILLEGAL_CELL, FALSE);
  header_assigned = TRUE;
  feature_offset = 0;
  bzero((void*)&hdr, sizeof(hdr));
  my_hdr.max_x_post = 2.0*ctdb_in->max_x_post;
  my_hdr.max_y_post = 2.0*ctdb_in->max_y_post;
  my_hdr.pages_wide = my_hdr.max_x_post/32 + 1;
  my_hdr.pages_high = my_hdr.max_y_post/32 + 1;
  hdr.pages_wide = my_hdr.pages_wide;
  hdr.pages_high = my_hdr.pages_high;
  ctdb_in -> pages_wide = my_hdr.pages_wide;
  ctdb_in -> pages_high = my_hdr.pages_high;
  ctdb_intern_pack_header(ctdb_in, &hdr);
  hdr.format = output_format;
  time((time_t *)&tloc);
```

This section initializes coordinate transformation routines.

This program uses only one cell.

My_hdr stores the information needed to increase the grid size for the output database, ctdb_out. In this particular case, the number of grid posts is doubled. The page variables specify how many pages (a 32 x 32 array of integers) are needed to store the final database. A patch is a 4x4 array of posts used to streamline feature lookup routines.

```
strcpy(hdr.date, ctime((time_t *)&tloc));
hdr.num_linear_models = CTDB_MAX_CANOPY_MODELS;
sprintf(hdr.name,options.output_base_name.value);
ctdb_intern_unpack_header(&ctdb_out, &hdr);
ctdb_out.elevations = elev_ptr;
ctdb_out.soils = soil_ptr;
my_hdr.min_x = 0.0; my_hdr.min_y = 0.0; my_hdr.min_z = 0.0;
my_hdr.max_x = ctdb_in->max_x;
my_hdr.max_y = ctdb_in->max_y;
my_hdr.max_z = ctdb_in->max_z;
my_hdr.patches_wide = (my_hdr.pages_wide*32 + 2)/4;
my_hdr.patches_high = (my_hdr.pages_high*32 + 2)/4;
my_hdr.incrD =
   (my_hdr.max_x - my_hdr.min_x)/
  (my_hdr.max_x_post + 0.0);
my_hdr.inv_incr = 1.0/my_hdr.incrD;
my_hdr.patch_incr = 4;
my_hdr.patch_inv_incr = 1.0/(float) (my_hdr.patch_incr);
my_hdr.patch_incrD = my_hdr.patch_incr;
my_hdr.incr = (float) (my_hdr.incrD + 0.5);
```

Note: my_hdr.incr is an integer whereas my_hdr.incrD is a float64. It is important that my_hdr. incrD is calculated from my_hdr. incr. Both variables are used by the ModSAF terrain routines.

```
min_patch_x = 0; min_patch_y = 0;
max_patch_x = my_hdr.patches_wide;
max_patch_y = my_hdr.patches_high;
ctdb_out.min_x = my_hdr.min_x;
ctdb_out.min_y = my_hdr.min_y;
ctdb_out.min_z = my_hdr.min_z;
ctdb_out.max_x = my_hdr.max_x;
ctdb_out.max_y = my_hdr.max_y;
ctdb_out.max_z = my_hdr.max_z;
ctdb_out.pages_wide = my_hdr.pages_wide;
ctdb_out.pages_high = my_hdr.pages_high;
ctdb_out.max_x_post = my_hdr.max_x_post;
ctdb_out.max_y_post = my_hdr.max_y_post;
ctdb_out.patches_wide = my_hdr.patches_wide;
ctdb_out.patches_high = my_hdr.patches_high;
ctdb_out.incr = my_hdr.incr;
ctdb_out.inv_incr = my_hdr.inv_incr;
ctdb_out.incrD = my_hdr.incrD;
ctdb_out.patch_incr = my_hdr.patch_incr;
ctdb_out.patch_inv_incr = my_hdr.patch_inv_incr;
ctdb_out.patch_incrD = my_hdr.patch_incrD;
```

These variables in the my_hdr structure are recalculated based on the array size specified by my_hdr.post_high and my_hdr.posts_wide. The structure for the output database, ctdb_out, is updated with the new information stored in the structure my_hdr.

```
ctdb_out.soil_tables = (int32 **)
   ctdb_alloc(NULL, ctdb_in->num_soil_tables *
   sizeof(int32 *),"Compiler rep. soil table");
ctdb_out.soil_table_storage = (int8 *)
   ctdb_alloc(NULL, ctdb_in->soil_table_size,"Compiler rep. soil table data");
bcopy((void*)ctdb_in->soil_table_storage,(void*)ctdb_out.soil_table_storage,
  ctdb_in->soil_table_size);
ctdb_out.soil_tables[0] =(int32 *)ctdb_out.soil_table_storage;
for(i=1;i<ctdb_in->num_soil_tables;i++)
   {
     ctdb_out.soil_tables[i] = (int32 *)(ctdb_out.soil_table_storage +
         ((int8 *)ctdb_in->soil_tables[i] - ctdb_in->soil_table_storage));
   }
ctdb_out.pat_table_ptr = (CTDB_PAT_TABLE_CMP *)
   ctdb_alloc(NULL, sizeof(CTDB_PAT_TABLE_CMP),
```

The soil table is copied from the old terrain database to the new database.

```
    "Compiler Polygon Attribute Table");
  ctdb_out.pat_table_ptr->num_columns = ctdb_in->pat_table_ptr->num_columns;
  ctdb_out.pat_table_ptr->alloced_columns = ctdb_in->pat_table_ptr->num_columns;
  ctdb_out.pat_table_ptr->num_entries = ctdb_in->pat_table_ptr->num_entries;
  ctdb_out.pat_table_ptr->alloced_entries = ctdb_in->pat_table_ptr->alloced_entries;
  ctdb_out.pat_table_ptr->columns = (CTDB_PAT_COLUMN_HDR_CMP *)
    ctdb_alloc(NULL, ctdb_out.pat_table_ptr->alloced_columns *
    sizeof(CTDB_PAT_COLUMN_HDR_CMP),
    "Compiler PAT column headers");
  for(i=0; i<ctdb_out.pat_table_ptr->num_columns; i++)
   {
     ctdb_out.pat_table_ptr->columns[i].facc =
      ctdb_in->pat_table_ptr->columns[i].facc;
     ctdb_out.pat_table_ptr->columns[i].data =
       (CTDB_PAT_DATA_CMP *)ctdb_alloc(NULL,
       ctdb_out.pat_table_ptr->alloced_entries *
       sizeof(CTDB_PAT_DATA_CMP), "Compiler PAT data");
     bcopy((void*)ctdb_in->pat_table_ptr->columns[i].data,
       (void*)ctdb_out.pat_table_ptr->columns[i].data,
       ctdb_in->pat_table_ptr->num_entries *
       sizeof(CTDB_PAT_DATA_CMP));
   }
  ctdb_out.water_chars = (CTDB_WATER_CHARS_CMP *)
    ctdb_alloc(NULL, ctdb_in->num_water_chars *
    sizeof(CTDB_WATER_CHARS_CMP),"Cmp. rep. water chars");
  bcopy((void*)ctdb_in->water_chars, (void*)ctdb_out.water_chars,
    ctdb_in->num_water_chars *sizeof(CTDB_WATER_CHARS_CMP));
 ctdb_out.water_chars[0].tidal_zone_index = 1;
 ctdb_out.tidal_zones = (CTDB_TIDAL_ZONE_CMP *)
    ctdb_alloc(NULL, ctdb_in->num_tidal_zones *
    sizeof(CTDB_TIDAL_ZONE_CMP),
    "Cmp. rep. water chars");
 bcopy((void*)ctdb_in->tidal_zones, (void*)ctdb_out.tidal_zones,
    ctdb_in->num_tidal_zones *sizeof(CTDB_TIDAL_ZONE_CMP));
 ctdb_out.volume_models = NULL;
 ctdb_out.mes_volume_models = NULL;
 ctdb_out.origin_x=-(ctdb_out.min_x+ctdb_out.max_x)/2;
 ctdb_out.origin_y=-(ctdb_out.min_y+ctdb_out.max_y)/2;
 ctdb_out.cell_id = GCS_ILLEGAL_CELL;
 ctdb_out.gcs_mode = gcs_mode;
 ctdb_out.west_long = indb_west_long;
 ctdb_out.south_lat = indb_south_lat;
 ctdb_out.east_long = indb_east_long;
 ctdb_out.north_lat = indb_north_lat;
 ctdb_out.elevations =transform_posts(ctdb_in->elevations);
  ctdb_in->pages_wide = saved_pages_wide;
  ctdb_in->pages_high = saved_pages_high;
 patch_water_state = (int32 *)
 ctdb_alloc(NULL, (ctdb_out.patches_wide)* (ctdb_out.patches_high)*
    sizeof(int32), "Patch water state array");
 bzero((void*)patch_water_state, (ctdb_out.patches_wide) *
    (ctdb_out.patches_high)* sizeof(int32));
 encode_abstract_features(patch_water_state);
 add_subcanopies (patch_water_state);
 hdr.num_features = encode_physical_features(output_format,
    dbname, dbase_type,&hdr.num_linear_models,
```

The polygonal pattern table is transferred from ctdb_in to ctdb_out.

The water characteristics and the tidal zone information are passed directly from ctdb_in to ctdb_out.

Transform_posts writes the new elevation posts to ctdb_out.

Encode_abstract features adds canopies and other polygonal features to the database. Encode_physical features adds roads, rivers, trees and buildings to the database.

```
    &hdr.num_aggregate_models, patch_water_state);
  ctdb_encode_abstract(&ctdb_out, &quad_root);
  hdr.num_quad_data = ctdb_out.num_quad_data;
  collect_nodes_and_edges(&hdr);
  ctdb_cmplr_write(&ctdb_out, dbname, output_format);
  ctdb_print_feature_stats();
  if (ctdb_out.features)
    munmap(ctdb_out.features,
     hdr.num_features * sizeof(CTDB_FEATURE_DATA_CMP));
  if (patch_water_state)
    STDDEALLOC(patch_water_state);
  }
  return 0;
}
```

A.3 Reorder_patch_features

static void reorder_patch_features (offsets, sizes, dbname, num_features)

Reoder_patch_features is unchanged from the original program recompile.c in the ModSAF 3.0 source code.

```
    int32 *offsets;
    int32 *sizes;
    char *dbname;
    int32 num_features;
    {
     FILE *fp;
     char temp_fname[255];
     int32 patch_x, patch_y, patch_number, offset = 0;
     sprintf(temp_fname, "%s.features",dbname);
      fwrite(ctdb_out.features, sizeof(CTDB_FEATURE_DATA_CMP),
     num_features, fp);
     fclose(fp);
    for(patch_y = 0; patch_y < ctdb_out.patches_high; patch_y++)
       {
         for(patch_x = 0; patch_x < ctdb_out.patches_wide; patch_x++)
           {
             patch_number = (patch_y * ctdb_out.patches_wide) + patch_x;
             if(offsets[patch_number] != NO_FEATURES)
               {
                 fseek(fp, offsets[patch_number] *
                  sizeof(CTDB_FEATURE_DATA_CMP),0);
                 fread(ctdb_out.features + offset,
                  sizeof(CTDB_FEATURE_DATA_CMP),
                 sizes[patch_number], fp);
                 ctdb_set_patch_feature_start(&ctdb_out, patch_number, offset);
                 offset += sizes[patch_number];
               }
         }
       }
     fclose(fp);
     unlink(temp_fname);
    }
```

A.4 Encode_physical_features

static int32 encode_physical_features (format, dbname, dbase_type, num_linear_models, num_aggregate_models, patch_water_state)

```
    int32 format;
```

```
char *dbname;
CTDB_DBASE_TYPE dbase_type;
int32 *num_linear_models;
int32 *num_aggregate_models;
int32 *patch_water_state;
{
 int32 patches;
 int32 patch_meters, old_patch_meters,new_patch_meters;
 int32 patch_x, patch_y, new_patch_x, new_patch_y;
 int32 old_patch_x, old_patch_y;
 int32 added = 0;
 CTDB_LAID_LINEAR *linear, *all_linear;
 char temp_fname[CTDB_FILENAME_DIMENSION_CMP];
 int32 i,j,gcs_patch_num;
 static CTDB_LAID_LINEAR local;
 CMP_REF tz_refs[CMP_MAX_REF];
 int8 *post_array = NULL;
 int32 *offsets, *sizes;
 trans_patches = (TRANSITION_PATCH_INFO *);
    ctdb_alloc(NULL,sizeof(TRANSITION_PATCH_INFO)*
    ctdb_out.patches_wide * ctdb_out.patches_high, "trans patches");
 bzero((void*)trans_patches, sizeof(TRANSITION_PATCH_INFO) *
   ctdb_out.patches_wide * ctdb_out.patches_high);
 offsets = (int32 *)ctdb_alloc(NULL, sizeof(int32) *
    ctdb_out.patches_wide *ctdb_out.patches_high, "offsets");
 for(i = 0; i < ctdb_out.patches_high * ctdb_out.patches_wide; i++)
    offsets[i] = NO_FEATURES;
sizes = (int32 *)ctdb_alloc(NULL, sizeof(int32) *
   ctdb_out.patches_wide *ctdb_out.patches_high, "sizes");
   bzero((void*)sizes, sizeof(int32) * ctdb_out.patches_wide *
   ctdb_out.patches_high);
 ctdb_zero_out_patch_buffer();
 sprintf(temp_fname, "%s.te",dbname);
 patches = ctdb_out.patches_wide * ctdb_out.patches_high;
 if(!ctdb_out.patch_groups)
    {
      ctdb_out.patch_groups =(CTDB_PATCH_GROUP_CMP *)STDALLOC(
        sizeof(CTDB_PATCH_GROUP_CMP)*
       (patches/CTDB_PATCHES_PER_GROUP_CMP));
     bzero((void*)ctdb_out.patch_groups,sizeof(CTDB_PATCH_GROUP_CMP)*
       (patches/CTDB_PATCHES_PER_GROUP_CMP));
    }
if(!ctdb_out.linear_models)
    {
      ctdb_out.linear_models = (CTDB_LINEAR_MODEL_CMP *)
       STDALLOC(MAX_NUM_LINEAR_MODELS *
        sizeof(CTDB_LINEAR_MODEL_CMP));
      bzero((void*)ctdb_out.linear_models, MAX_NUM_LINEAR_MODELS *
       sizeof(CTDB_LINEAR_MODEL_CMP));
    }
if(!ctdb_out.aggregate_models)
    {
     ctdb_out.aggregate_models = (CTDB_AGGREGATE_MODEL_CMP **)
       STDALLOC(MAX_NUM_AGGREGATE_MODELS *
       sizeof(CTDB_AGGREGATE_MODEL_CMP *));
     bzero((void*)ctdb_out.aggregate_models,
```

```
            MAX_NUM_AGGREGATE_MODELS *
            sizeof(CTDB_AGGREGATE_MODEL_CMP *));
        }
    old_patch_meters = ctdb_in->incr * ctdb_in->patch_incr;
    new_patch_meters = ctdb_out.incr * ctdb_out.patch_incr;
    for (new_patch_y = 0; new_patch_y < ctdb_out.patches_high; new_patch_y++, putchar('.'),fflush(stdout))
        {
        for (new_patch_x = 0; new_patch_x < ctdb_out.patches_wide; new_patch_x++)
          {
            old_patch_x = xx/old_patch_meters;
            old_patch_y = yy/old_patch_meters;
            patch_x = old_patch_x;
            patch_y = old_patch_y;
            gcs_patch_num = patch_y * ctdb_in->patches_wide + patch_x;
            ctdb_init_patch_buffer();
            collect_canopies(new_patch_x,new_patch_y,patch_x,patch_y);
            collect_buildings (new_patch_x,new_patch_y, old_patch_x,old_patch_y);
            collect_trees (new_patch_x,new_patch_y, old_patch_x,old_patch_y);
            collect_linears (new_patch_x,new_patch_y, old_patch_x,old_patch_y);
            if (( old_patch_y < ctdb_in->patches_high) &&  ( old_patch_x > 0))
             {
                collect_buildings (new_patch_x,new_patch_y, old_patch_x-1,old_patch_y+1);
                collect_trees (new_patch_x,new_patch_y, old_patch_x-1,old_patch_y+1);
                collect_linears(new_patch_x,new_patch_y, old_patch_x-1,old_patch_y+1);
             }
            if ( old_patch_y < ctdb_in->patches_high)
             {
                collect_buildings (new_patch_x,new_patch_y, old_patch_x,old_patch_y+1);
                collect_trees (new_patch_x,new_patch_y, old_patch_x,old_patch_y+1);
                collect_linears (new_patch_x,new_patch_y, old_patch_x,old_patch_y+1);
             }
            if (( old_patch_y < ctdb_in->patches_high) &&  ( old_patch_x < ctdb_in->patches_wide))
             {
                collect_buildings (new_patch_x,new_patch_y, old_patch_x+1,old_patch_y+1);
                collect_trees (new_patch_x,new_patch_y, old_patch_x+1,old_patch_y+1)
                collect_linear (new_patch_x,new_patch_y, old_patch_x+1,old_patch_y+1);
             }
            if ( old_patch_x < ctdb_in->patches_wide)
             {
                collect_buildings (new_patch_x,new_patch_y, old_patch_x+1,old_patch_y);
                collect_trees(new_patch_x,new_patch_y, old_patch_x+1,old_patch_y);
                collect_linear (new_patch_x,new_patch_y, old_patch_x+1,old_patch_y);
             }
            if (( old_patch_y > 0) &&  ( old_patch_x < ctdb_in->patches_wide))
             {
                collect_buildings (new_patch_x,new_patch_y, old_patch_x+1,old_patch_y-1);
                collect_trees (new_patch_x,new_patch_y, old_patch_x+1,old_patch_y-1);
                collect_linears (new_patch_x,new_patch_y, old_patch_x+1,old_patch_y-1);
             }
            if ( old_patch_y > 0)
             {
                collect_buildings (new_patch_x,new_patch_y, old_patch_x,old_patch_y-1);
                collect_trees (new_patch_x,new_patch_y, old_patch_x,old_patch_y-1);
                collect_linears (new_patch_x,new_patch_y, old_patch_x,old_patch_y-1);
             }
            if (( old_patch_x > 0) && ( old_patch_y > 0)
```

The new patches may overlap more than one of the original patches, so features must be collected for each of the nine original patches that might intersect the new patch.

```
            {
                collect_buildings (new_patch_x,new_patch_y, old_patch_x-1,old_patch_y - 1);
                collect_trees (new_patch_x,new_patch_y, old_patch_x-1,old_patch_y - 1);
                collect_linears(new_patch_x,new_patch_y, old_patch_x-1,old_patch_y - 1);
            }
            if ( old_patch_x > 0)
            {
                collect_buildings (new_patch_x,new_patch_y, old_patch_x-1,old_patch_y);
                collect_trees (new_patch_x,new_patch_y, old_patch_x-1,old_patch_y);
                collect_linears(new_patch_x,new_patch_y, old_patch_x-1,old_patch_y);
            }
            added = ctdb_store_patch_buffer(&ctdb_out,
                new_patch_x + offset_x, new_patch_y + offset_y,
                ctdb_out.features+feature_offset, ctdb_out.linear_models,
                &ctdb_out.num_linear_models, ctdb_out.aggregate_models,
                &ctdb_out.num_aggregate_models,complete_micro_poly,
                &num_te, trans_patches, &trans_end,
                &max_elev, format, dbase_type, FALSE,
                FALSE, fptr, FALSE);
            if(added)
            {
                offsets[(new_patch_y + offset_y) * ctdb_out.patches_wide +
                    new_patch_x + offset_x] = feature_offset;
                sizes[(new_patch_y + offset_y) * ctdb_out.patches_wide +
                    new_patch_x + offset_x] = added;
            }
            feature_offset += added;
        }
    }
    if(ctdb_in->features) STDDEALLOC(ctdb_in->features);
    if (ctdb_in->elevations) STDDEALLOC(ctdb_in->elevations);
    if (ctdb_in->soils) STDDEALLOC(ctdb_in->soils);
    if (post_array) STDDEALLOC(post_array);
    reorder_patch_features(offsets, sizes, dbname, feature_offset);
    STDDEALLOC(offsets);
    STDDEALLOC(sizes);
    fclose(fptr);
    ctdb_out.n_te_bytes = total_tes * sizeof(TERRAIN_ELEMENT_16);
    *num_linear_models = ctdb_out.num_linear_models;
    *num_aggregate_models = ctdb_out.num_aggregate_models;
    return feature_offset;
}
```

Ctdb_store_patch_buffer writes all the physical features to ctdb_out.

A.5 Look_for_features

```
static void look_for_features (patch_water_state)
    int32 *patch_water_state;
    {
    CTDB_SEARCH_SPACE_PTR_CMP search_space;
    int32 i, j,ix,iy,ii,max_steps,flag;
    int32 count,code, data_size;
    float32 *verts, *lverts = NULL;
    float32 x1,y1,x2,y2,cx,cy,delx,dely,slope,step,my_sign;
    CTDB_ABSTRACT_DATA_CMP *data;
    I int32 num_lverts = 0, *soil_table,soil;
    float64 meters_per_patch;
    meters_per_patch = ctdb_in->incrD * ctdb_in->patch_incrD;
```

```
step = ctdb_in->incrD/10.0;
search_space =
   ctdb_intern_create_search_space(ctdb_in, ctdb_in->min_x, ctdb_in->min_y,
     ctdb_in->max_x, ctdb_in->max_y, CTDB_BY_PATCH);
while (ctdb_intern_next_quad_patch(search_space))
   {
     for (code=1;code<=24;code++)
       while (count = ctdb_intern_next_abstract(search_space, code, &verts, &data, &data_size))
       {
         num_lverts = count;
         if (code == CTDB_ABSTRACT_CANOPY_CMP)
              {
                  for (i=2;i<(2*count);i+=2)
                   {
                      x1 = verts[i-2]; y1 = verts[i-1];
                      x2 = verts[i]; y2 = verts[i+1];
                      delx = x2-x1; dely = y2-y1;
                      if ((delx*delx) > (dely*dely))
                      {
                        slope = dely/delx;
                        my_sign = 1.0;
                        if (delx < 0) my_sign = -1.0;
                        max_steps = sqrt(delx*delx)/(2.0*mat_step);
                        for (ii = 0; ii < max_steps;ii++)
                          {
                             cx = x1 + ii*step*my_sign;
                             cy = slope*(cx - x1) + y1;
                             ix = (float) (cx/mat_step);
                             y = (float) (cy/mat_step);
                             canopy[ix][iy] = 1;
                          }
                      }
                      else
                      {
                        slope = delx/dely;
                        my_sign = 1.0;
                        if (dely < 0) my_sign = -1.0;
                        max_steps = sqrt(dely*dely)/(2.0*mat_step);
                        for (ii = 0; ii < max_steps;ii++)
                        {
                          cy = y1 + ii*step*my_sign;
                          cx = slope*(cy - y1) + x1;
                          ix = (float) (cx/mat_step);
                          iy = (float) (cy/mat_step);
                          canopy[ix][iy] = 1;
                        }
                   }
              }
         }
       }
   }
for (iy = 0;iy < 5500;iy++)
   {
     flag = 0;
     if ( canopy[0][iy] == 1) flag = 1;
     for (ix = 1; ix < 5500;ix++)
```

Look_for_features finds all the canopies and lakes on the original database so that the soil type for the underlying posts can be adjusted. The array canopy is a high resolution map of the location of the canopies.

```
            {
              if ((canopy[ix][iy] == 0) && (flag == 1))
                  canopy[ix][iy] = 2;
              if ((canopy[ix][iy] == 1) && (flag == 0))
                  flag = 1;
              else if ((canopy[ix][iy] == 1) && (flag == 1))
                  flag = 0;
            }
        }
    ctdb_intern_destroy_search_space(search_space);
    if (num_lverts > 0) free(lverts);
  }
```

A.6 Encode_abstract_features

```
static void encode_abstract_features (patch_water_state)
    int32 *patch_water_state;
    {
     CTDB_SEARCH_SPACE_PTR_CMP search_space;
     int32 i, j;
     int32 count;
     int32 code;
     float32 *verts;
     CTDB_ABSTRACT_DATA_CMP *data;
     int32 data_size;
     float32 *lverts = NULL;
     int32 num_lverts = 0;
     int32 *soil_table;
     int32 soil;
     float64 meters_per_patch;
     meters_per_patch = ctdb_out.incrD * ctdb_out.patch_incrD;
     search_space = ctdb_intern_create_search_space(ctdb_in,
       ctdb_out.min_x, ctdb_out.min_y,
       ctdb_out.max_x, ctdb_out.max_y,
       ctdb_BY_PATCH);
    while (ctdb_intern_next_quad_patch(search_space))
     {
       for (code=1;code<CTDB_ABSTRACT_MAX;code++)
       while (count = ctdb_intern_next_abstract(search_space, code,&verts, &data, &data_size))
          {
            if (count > num_lverts)
            {
             lverts = (float32*)STDREALLOC(lverts,2*count*sizeof(float32));
             num_lverts = count;
            }
            for(i=0;i<(2*count);i+=2)
            {
             lverts[i] = verts[i] + offset_x_meters;
             lverts[i+1] = verts[i+1] + offset_y_meters;
            }
           if (code == CTDB_ABSTRACT_CANOPY_CMP)
           data->canopy.impenetrable = 1;
           ctdb_store_abstract(&ctdb_out, &quad_root,code, count, lverts,
               CTDB_ABSTRACT_DATA_SIZE(data_size),
               (int32 *)data, min_patch_x * meters_per_patch,
               min_patch_y * meters_per_patch,max_patch_x * meters_per_patch,
                max_patch_y * meters_per_patch);
```

Encode_abstract tranfers the data for lakes, canopies, and other abstract features from ctdb_in to ctdb_out. Note that the polygons do not depend on patch size so they do not need to be adjusted for the new database.

```
        }
    }
    ctdb_intern_destroy_search_space(search_space);
    if (num_lverts > 0) free(lverts);
  }
```

A.7 Collect_nodes_and_edges

```
static void collect_nodes_and_edges(hdr)
    CTDB_FILE_HEADER *hdr;
  {
    ctdb_create_networks(&ctdb_out, output_format);
    hdr->num_nodes = ctdb_out.num_nodes;
    hdr->node_size = ctdb_out.node_size;
    hdr->num_edges = ctdb_out.num_edges;
    hdr->edge_size = ctdb_out.edge_size;
  }
```

Collect_nodes_and_edges is copied from the original program recompile.c in the ModSAF 3.0 source code.

A.8 Transform_posts

```
static int32 *transform_posts()
  {
    float64 x,y,dx,dy,zz,z00,z10,z01,z11;
    float64 xd,yd,loc_x, loc_y;
    float64 page_meters, post_meters,inv_post_meters;
    uint32 p, s;
    int32 *new_posts, *new_soils;
    int32 norm_x, norm_y ,ix,iy;
    int32 post_index,page, index;
    new_posts = (int32 *)ctdb_alloc(NULL, ctdb_in->pages_wide *
       ctdb_in->pages_high * PAGE_SIZE,"New elevation posts");
    new_soils = (int32 *)ctdb_alloc(NULL, ctdb_in->pages_wide *
       ctdb_in->pages_high* PAGE_SIZE, "New soil indices");
    post_meters = ctdb_out.incrD;
    page_meters = POSTS_PER_SIDE * post_meters;
    inv_post_meters = 1.0/post_meters;
    add_hills_to_patches();
    for (y=0; y<=ctdb_out.max_y; y+=page_meters)
       for (x=0; x<=ctdb_out.max_x; x+=page_meters)
         for (dy=0.0; dy<page_meters; dy+=post_meters)
           for (dx=0.0; dx<page_meters; dx+=post_meters)
             {
              loc_x = x+dx;
              loc_y = y+dy;
              ix = (float) (loc_x/ctdb_in->incrD);
              iy = (float) (loc_y/ctdb_in->incrD);
              xd = (loc_x - (float) (ix*ctdb_in->incrD));
              yd = (loc_y - (float) (iy*ctdb_in->incrD));
              z00 = xd*(original[ix +1][iy] - original[ix][iy])/ctdb_in->incrD+
                  yd*(original[ix][iy + 1] - original[ix][iy])/ctdb_in->incrD + original[ix][iy];
              z11 = xd* (original[ix][iy+1] - original[ix+1][iy+1])/ctdb_in->incrD+ yd*
                  (original[ix+1][iy] - original[ix+1][iy+1])/ctdb_in->incrD + original[ix+1][iy+1];
              norm_x = (int32)POST_X(loc_x * inv_post_meters);
              norm_y = (int32)POST_Y(loc_y * inv_post_meters);
              zz = (z00 + z11)/2.0;
              zz = zz + add_vrt(loc_x,_locy);
```

Add_hills_to_patches adds vrt hills to each patch so that the original elevation data are not altered. Transform_posts produces the elevation grid for ctdb_out from the original elevation grid stored in original and the new vrt hills. This routine also adjusts the soil type, based on the array my_soil.

```
            p = BUILD_POST_ELEVATION (zz, ctdb_in->inv_fixed_point_basis);
            p &= ~POST_FLAGS;
            if (canopy[ix][iy] == 1) p |= POST_TREES;
            s = my_soil[temp_x][temp_y];
            post_index = ctdb_intern_get_post_index(ctdb_in, norm_x, norm_y);
            new_posts[post_index] = p;
            new_soils[post_index] = s;
          }
    ctdb_out.soils = new_soils;
    return new_posts;
  }
```

A.9 Add_hills_to_patches

add_hills_to_patches();

```
  {
    int ix,iy,ih;
    float64 w,base_hgt,xx,yy,x0,y0,x1,y1;
    float64 d,d00,d01,d11,d10;
    w = ctdb_in->patch_width;
    base_hgt = 5.0;
    for (iy =0;iy< ctdb_in->patches_high;iy++)
      {
        for (ix=0;ix < ctdb_in->patches_wide;ix++)
          {
            hills_per_patch[ix][iy] = 5.0*drand48();
            for(ih=0;ih < hills_per_patch[ix][iy];ih++)
             {
                hill[ihill].cx = xx= w*drand48();
                hill[ihill].cy = yy= w*drand48();
                hill[ihill].hgt = base_hgt*drand48();
                hill[ihill].angle = 180.0*drand48();
                hill[ihill].wgty = 0.75*drand48() + 0.25;
                hill[ihill].power = 5.0*drand48() + 0.5;
                x0 = (int) (hill[ihill].cx/ctdb_in->incrD);
                y0 = (int) (hill[ihill].cy/ctdb_in->incrD);
                x1 = x0 + ctdb_in->incrD;
                y1 = y0 + ctdb_in->incrD;
                d00 = sqrt((xx-x0)*(xx-x0) +(yy-y0)*(yy-y0));
                d10 = sqrt((xx-x1)*(xx-x1) +(yy-y0)*(yy-y0));
                d11 = sqrt((xx-x1)*(xx-x1) +(yy-y1)*(yy-y1));
                d01= sqrt((xx-x0)*(xx-x0) +(yy-y1)*(yy-y1));
                d = d00;
                if (d > d10) d = d10;
                if (d > d11) d = d11;
                if (d > d01) d = d01;
                hill[ihill].wgtx = d*drand48() + 0.5;
                hill[ihill].cx = hill[ihill].cx + ctdb_in->patch_width;
                hill[ihill].cy = hill[ihill].cy + ctdb_in->patch_width;
             }
          }
       }
     }
  }
```

A.10 Complete_micro_poly

static void complete_micro_poly(poly)

Complete_micro_poly is copied from the original program recompile.c in the ModSAF 3.0 source code.

```
CTDB_MC_POLYGON *poly;
{
 int32 i;
 int32 norm_x, norm_y;
 float64 midx, midy;
 midx = midy = 0.0;
 for (i=0;i<3;i++)
    {
     norm_x = poly->verts[i][X] * ctdb_out.inv_incr;
     norm_y = poly->verts[i][Y] * ctdb_out.inv_incr;
     if (gcs_mode == CTDB_MODE_SIMNET)
       {
          poly->verts[i][Z] =ctdb_intern_lookup_post_elevation(ctdb_in, norm_x, norm_y);
       }
       else
       {
          poly->verts[i][Z] =ctdb_intern_lookup_post_elevation(&ctdb_out, norm_x, norm_y);
       }
     midx += norm_x;
     midy += norm_y;
    }
 midx /= 3.0;
 midy /= 3.0;
 poly->soil = ctdb_intern_lookup_soil(ctdb_in, midx, midy);
}
```

A.11 Collect_building

```
static void collect_building (new_patch_x,new_patch_y,old_patch_x,old_patch_y)
    int32 new_patch_x,new_patch_y, old_patch_x,old_patch_y;
    {
    CTDB_VOLUME_POLYGON building;
    int32 n_verts, i, j, offset;
    CTDB_FEATURE_DATA_CMP *feature, *last;
    int32 patch_number = old_patch_x + old_patch_y * ctdb_in->patches_wide;
    int32 next, fclass,x,y;
    int32 meters_per_patch = ctdb_in->incr * ctdb_in->patch_incr;
    int32 patch_min_x = old_patch_x * meters_per_patch;
    int32 patch_min_y = old_patch_y * meters_per_patch;
    int32 patch_max_x = (old_patch_x+1) * meters_per_patch;
    int32 patch_max_y = (old_patch_y+1) * meters_per_patch;
    int32 intersection;
    offset = 3;
    feature = ctdb_intern_lookup_patch(ctdb_in, patch_number);
    if (feature)
     {
       last = feature + feature->hdr.size;
       feature++;
       for(; feature < last; feature += next)
         {
          n_verts = feature->info.vertices;
          fclass = feature->info.feature_class;
           if (fclass == FC_MICRO_CMP)
             {next = NEXT_MICRO_CMP(n_verts);continue; }
          else if (fclass == FC_LINEAR_CMP)
             {next = NEXT_LINEAR_CMP(n_verts);continue; }
          else if (fclass == FC_CANOPY_CMP)
```

This section skips all features except the buildings

```
      {next = NEXT_CANOPY_CMP(n_verts);continue; }
    else if (fclass == FC_AGGREGATE_CMP)
      {next = NEXT_AGGREGATE_CMP(n_verts);continue; }
    else if (fclass == FC_LAID_LINEAR_CMP)
      {next = NEXT_LAID_LINEAR(n_verts); continue; }
    else if (fclass == FC_VOLUME_CMP)
      next = NEXT_VOLUME_CMP(n_verts);
    if (n_verts)
      {
       recompile_buildings++;
       building.n_verts = n_verts;
       for (i=0;i<n_verts;i++)
          {
            building.verts[i][X] = ctdb_in->incrD *
               XYO_TO_POST(feature[offset+2*i].xy.x,old_patch_x,
               ctdb_in->patch_incrD) + offset_x_meters;
            building.verts[i][Y] = ctdb_in->incrD *
               XYO_TO_POST(feature[offset+2*i].xy.y,old_patch_y,
               ctdb_in->patch_incrD) + offset_y_meters;
            building.verts[i][Z] = feature[offset+1+2*i].z;
          }
     if ( n_verts < 3)
          {
            building.n_verts = 3;
            building.verts[2][X] = 10.0 + ctdb_in->incrD *
               XYO_TO_POST(feature[offset].xy.x,old_patch_x,
               ctdb_in->patch_incrD) + offset_x_meters;
            building.verts[2][Y] = ctdb_in->incrD *
               XYO_TO_POST(feature[offset].xy.y,old_patch_y,
               ctdb_in->patch_incrD) + offset_y_meters;
            building.verts[2][Z] = feature[offset+1].z;
          }
    building.type = feature[1].volume.type;
    building.reference = feature[1].volume.reference;
    ctdb_buffer_vol(&building);
    }
  }
 }
}
```

Volume features are sets of polygon vertices in three-dimensional space. For each voulme feature, building stores those vertices that intersect the current patch. If there are fewer than three vertices, an extra vertex is added to complete the polygon.

A.12 Collect_trees

```
static void collect_trees (new_patch_x,new_patch_y,old_patch_x,old_patch_y)
   int32 new_patch_x, new_patch_y,old_patch,old_patch_y;
   {
    CTDB_LINEAR tree;
    int32 n_verts, i,ix,iy;
    CTDB_FEATURE_DATA_CMP *feature, *last;
    int32 patch_number = old_patch_x + old_patch_y * ctdb_in->patches_wide;
    int32 next, fclass;
    int32 my_max_x, my_max_y, my_min_x, my_min_y;
    float64 xx,yy,meters_per_patch;
    my_min_x = new_patch_x*ctdb_out.patch_incr*ctdb_out.incr;
    my_min_y = new_patch_y*ctdb_out.patch_incr*ctdb_out.incr;
    my_max_x = (new_patch_x + 1)*ctdb_out.patch_incr*ctdb_out.incr;
    my_max_y = (new_patch_y + 1)*ctdb_out.patch_incr*ctdb_out.incr;
    meters_per_patch = ctdb_out.incrD * ctdb_out.patch_incrD;
```

```
feature = ctdb_intern_lookup_patch(ctdb_in, patch_number);
if (feature)
  {
   last = feature + feature->hdr.size;
   feature++;
   for(; feature < last; feature += next)
     {
      n_verts = feature->info.vertices;
      fclass = feature->info.feature_class;
      if (fclass == FC_MICRO_CMP)
         {next = NEXT_MICRO_CMP(n_verts);continue; }
      else if (fclass == FC_VOLUME_CMP)
         {next = NEXT_VOLUME_CMP(n_verts);continue; }
      else if (fclass == FC_CANOPY_CMP)
         {next = NEXT_CANOPY_CMP(n_verts);continue; }
      else if (fclass == FC_AGGREGATE_CMP)
         {next = NEXT_AGGREGATE_CMP(n_verts);continue;}
     else if (fclass == FC_LAID_LINEAR_CMP)
         {next = NEXT_LAID_LINEAR_CMP(n_verts);continue; }
     else if (fclass == FC_LINEAR_CMP)
         next = NEXT_LINEAR_CMP(n_verts);
     if (n_verts)
      {
         recompile_trees++;
         tree.n_verts = n_verts;
         tree.desc.type = CTDB_FT_TREELINE_CMP;
          tree.desc.linear_data.treeline_info.foliage_height =
          ctdb_in->incrD*XYO_TO_POST(ctdb_in->linear_models[
             feature[1].linear.reference].linear_data.treeline_info.
             foliage_height, 0, ctdb_in->patch_incrD);
         tree.desc.linear_data.treeline_info.trunk_radius =
             ctdb_in->incrD*XYO_TO_POST(ctdb_in->linear_models[
             feature[1].linear.reference].linear_data.treeline_info.
              trunk_radius, 0, ctdb_in->patch_incrD);
          tree.desc.width = ctdb_out.incrD * XYO_TO_POST(
             ctdb_in->linear_models[feature[1].linear.reference].width,
             0, ctdb_in->patch_incrD);

         tree.desc.linear_data.treeline_info.fullness =
             ctdb_in->linear_models[feature[1].linear.reference].
              linear_data.treeline_info.fullness /
             (float32)(1 << CTDB_FULLNESS_BITS_CMP - 1);
         tree.desc.linear_data.treeline_info.total_height = 0.0;
         for (i=0;i<n_verts;i++)
         {
             tree.verts[i][X] = ctdb_in->incrD *
                XYO_TO_POST(feature[2+2*i].xy.x,old_patch_x,
                ctdb_in->patch_incrD) + offset_x_meters;
             tree.verts[i][Y] = ctdb_in->incrD *
                XYO_TO_POST(feature[2+2*i].xy.y,old_patch_y,
                ctdb_in->patch_incrD) + offset_y_meters;
             tree.verts[i][Z] = feature[3+2*i].z;
         }

      ctdb_buffer_linear(&tree);
      if (n_verts =1)
```

This section skips all features except the tree lines and individual trees.

Tree lines are linear segments with information about foliage height, foliage density, tree spacing, and trunk radius.

Individual trees are stored for the first and last tree in a tree line.

```
            {
              tree.n_verts = 1;
               tree.verts[1][X] = tree.verts[0][X] + 20.0;
               tree.verts[1][Y] = tree.verts[0][Y];
              ctdb_buffer_linear(&tree);
            }
          if ((n_verts > 1) && (feature[1].integer & M_TREE_FIRST_CMP))
            {
              ctdb_buffer_linear(&tree);
              tree.n_verts = 1;
              ctdb_buffer_linear(&tree);
            }
          if ((n_verts > 1) && (feature[1].integer & M_TREE_LAST_CMP))
            {
              tree.n_verts = 1;
              tree.verts[0][X] = tree.verts[n_verts-1][X] +offset_x_meters;
              tree.verts[0][Y] = tree.verts[n_verts-1][Y] +offset_y_meters;
              tree.verts[0][Z] = tree.verts[n_verts-1][Z];
              ctdb_buffer_linear(&tree);
            }
          }
        }
      }
    }
```

A.13 Collect_linear

```
static void collect_linear (new_patch_x,new_patch_y,old_patch_x,old_patch_y)
    int32 new_patch_x,new_patch_y,old_patch_x,old_patch_y;
    {
     CTDB_LAID_LINEAR line;
     float64 verts[CTDB_VERTICES_MAX][2];
     float64 width,xx,yy,x1,y1,x2,y2;
     float64 xm,ym,xM,yM,del_x, del_y;
     int32 soil,n_verts,,k,j;
     CTDB_FEATURE_DATA_CMP *feature, *last;
     int32 patch_number = old_patch_x + old_patch_y * ctdb_in->patches_wide;
     int32 next, fclass;
     feature = ctdb_intern_lookup_patch(ctdb_in, patch_number);
     if (feature)
       {
        last = feature + feature->hdr.size;
        feature++;
        for(; feature < last; feature += next)
          {
           n_verts = feature->info.vertices;
           fclass = feature->info.feature_class;
           if (fclass == FC_MICRO_CMP)
              {next = NEXT_MICRO_CMP(n_verts);continue; }
           else if (fclass == FC_VOLUME_CMP)
              {next = NEXT_VOLUME_CMP(n_verts);continue; }
          else if (fclass == FC_CANOPY_CMP)
              {next = NEXT_CANOPY_CMP(n_verts);continue; }
          else if (fclass == FC_AGGREGATE_CMP)
              {next = NEXT_AGGREGATE_CMP(n_verts);continue; }
          else if (fclass == FC_LINEAR_CMP)
              {next = NEXT_LINEAR_CMP(n_verts);continue; }
          else if (fclass == FC_LAID_LINEAR_CMP)
```

This section skips all features except the roads, rivers, and other linear features.

```
          next = NEXT_LAID_LINEAR_CMP(n_verts);
        if (n_verts)
        {
         if (n_verts >= MAX_LAID_LINEAR_VERTS)
          {
            printf("MAX_LAID_LINEAR_VERTS (ct_cmplr.h) exceeded. Increase it to %d\n",
            n_verts+1);
             n_verts = MAX_LAID_LINEAR_VERTS-1;
          }
         recompile_linears++;
         line.n_verts = n_verts;
         line.width = ctdb_in->incrD *
           XYO_TO_POST(feature[2].laid_linear.width,0,
           ctdb_in->patch_incrD);
         line.soil = feature[1].ll_soil.poly_char_index;
          if (line.soil == 1 ) line.width = 10.0;
          else line.width = 20.0;
          for (i=0;i<n_verts;i++)
           {
             xx = ctdb_in->incrD *
               XYO_TO_POST(feature[3+i].xy.x,old_patch_x,
               ctdb_in->patch_incrD) + offset_x_meters;
             yy = ctdb_in->incrD *
               XYO_TO_POST(feature[3+i].xy.y,old_patch_y,
               ctdb_in->patch_incrD) + offset_y_meters;
             line.verts[i][X] = xx;
             line.verts[i][Y] = yy;
           }
         ctdb_buffer_laid_linear(&line);
         }
      }
    }
}
```

Roads and rivers are line segments having a soil type and a width.

A.14 Conopy_triangles

```
static void canopy_triangle (micro,x0,y0,x1,y1,x2,y2)
    float64 x0,y0,x1,y1,x2,y2;
    CTDB_MC_POLYGON micro;
     {
      micro.n_verts = 3;
      micro.soil = 720907;
      micro.reserved = 0;
      micro.verts[0][X] = x0;
      micro.verts[0][Y] = y0;
      micro.verts[0][Z] = 20.0;
      micro.verts[1][X] = x1
      micro.verts[1][Y] = y1;
      micro.verts[1][Z] = 20.0;
      micro.verts[2][X] = x2
      micro.verts[2][Y] = y2;
      micro.verts[2][Z] = 20.0;
     }
```

A.15 Collect_canopies

```
static void collect_canopies  ( new_patch_x,new_patch_y,patch_x,patch_y,
```

```
n_cpoly, canopy_polys, n_cedge, canopy_edges,cp_limit, ce_limit)
   int32 patch_x,patch_y,new_patch_x, new_patch_y;
   int32 *n_cpoly;
   CTDB_MC_POLYGON canopy_polys[];
   int32 *n_cedge;
   CTDB_CAN_EDGE canopy_edges[];
   int32 cp_limit, ce_limit;
  {
    int32 patch_number, reference;
    CTDB_FEATURE_DATA_CMP *feature, *last, *next;
    CTDB_MC_POLYGON micro;
    CTDB_CAN_EDGE edge;
    int32 verts, fclass;
    int ix,iy,ix_start,ix_stop,iy_start,iy_stop,my_flag;
    float64 x1,y1,x2,y2,d1,d2,d3;
    float64 min_x,min_y,max_x,max_y;
    int zero,one,two;
    patch_number = patch_x + patch_y * ctdb_in->patches_wide;
    feature = ctdb_intern_lookup_patch(ctdb_in, patch_number);
    min_x = new_patch_x*ctdb_out.patch_incr*ctdb_out.incr;
    min_y = new_patch_y*ctdb_out.patch_incr*ctdb_out.incr;
    max_x = (new_patch_x + 1)*ctdb_out.patch_incr*ctdb_out.incr;
    max_y = (new_patch_y + 1)*ctdb_out.patch_incr*ctdb_out.incr;
    ix_start = min_x/mat_step; iy_start = min_y/mat_step;
    ix_stop = max_x/mat_step; iy_stop = max_y/mat_step;
     zero = 0; one = 0; two = 0;
    verts = 3;
    fclass = FC_CANOPY_CMP;
     for (iy = iy_start; iy < iy_stop; iy++)
       {
        for (ix = ix_start; ix < ix_stop; ix++)
          {
            if (canopy[ix][iy] == 0 ) zero = zero + 1;
            if (canopy[ix][iy] == 1 ) one = one + 1;
            if (canopy[ix][iy] == 2 ) two = two + 1;
          }
       }
     if ( (zero == 0) && (one == 0))
       {
        canopy_triangle(&micro,min_x,min_y,max_x,min_y,max_x,max_y);
        ctdb_buffer_canopy(&micro);
        canopy_triangle(&micro,min_x,min_y,min_x,max_y,max_x,max_y);
        ctdb_buffer_canopy(&micro);
       }
    if ( (two > 0) && (one > 0))
       {
        reference = 0;
        edge.fullness = DEFAULT_FULLNESS;
        edge.start[X] = min_x; edge.start[Y] = min_y;
        edge.end[X] = max_x; edge.end[Y] = max_y;
        ctdb_buffer_cedge(&edge);
        if ( canopy[ix_start][iy_stop-1] == 0 && canopy[ix_start][iy_start] == 0
          && canopy[ix_stop-1][iy_start] == 0  && canopy[ix_stop-1][iy_stop-1] == 0 )
          {
          if ( canopy[(ix_stop-1 + ix_start)/2][iy_start ] == 2
             && canopy[(ix_stop-1+ ix_start)/2][iy_stop-1] == 2
```

```
        && canopy[ix_start][(iy_stop-1 + iy_start) /2] == 0
        && canopy[ix_stop-1][ (iy_stop-1 + iy_start)/2] == 0 )
        {
        canopy_triangle(&micro,
            (max_x + min_x) / 4 + (min_x / 2),min_y,
            (max_x + min_x) / 4 + (min_x / 2),min_y,
            (max_x + min_x) / 4 + (min_x / 2),max_y);
         ctdb_buffer_canopy(&micro);
         canopy_triangle(&micro,
            (max_x + min_x) / 4 + (min_x / 2),max_y,
            (max_x + min_x) / 4 + (min_x / 2),min_y,
            (max_x + min_x) / 4 + (min_x / 2),max_y);
         ctdb_buffer_canopy(&micro);
        }
    else
        {
         canopy_triangle(&micro,
            (max_x + min_x) / 2, (max_y + min_y) / 2,
            min_x, (max_y + min_y) / 2,
            (max_x + min_x) / 2,max_y);
         ctdb_buffer_canopy(&micro);
         canopy_triangle(&micro,
            (max_x + min_x) / 2, (max_y + min_y) / 2,
            (max_x + min_x) / 2, max_y
            max_x,(max_y + min_y) / 2);
         ctdb_buffer_canopy(&micro);
         canopy_triangle(&micro,
            (max_x + min_x) / 2, (max_y + min_y) / 2,
            (max_x + min_x) / 2, min_y
            min_x,(max_y + min_y) / 2);
         ctdb_buffer_canopy(&micro);
         canopy_triangle(&micro,
            (max_x + min_x) / 2, (max_y + min_y) / 2,
            max_x,(max_y + min_y) / 2
            (max_x + min_x) / 2,min_y);
         ctdb_buffer_canopy(&micro);
        }
   }
   else if ( canopy[ix_start][iy_stop-1] == 0  && canopy[ix_stop-1][iy_start] == 2 )
     {
     reference = 0;
     edge.fullness = DEFAULT_FULLNESS;
     edge.start[X] = max_x; edge.start[Y] = min_y;
     edge.end[X] = max_x; edge.end[Y] = max_y;
     ctdb_buffer_cedge(&edge);
     canopy_triangle(&micro, min_x, min_y,  max_x,min_y, max_x,max_y);
         ctdb_buffer_canopy(&micro);
     if ( canopy[ix_start][(iy_stop-1 + iy_start) / 2] == 2 )
        {
         canopy_triangle(&micro,  min_x, min_y,
            min_x, (max_y + min_y) / 2,
            (max_x + min_x) / 2, (max_y + min_y) / 2);
         ctdb_buffer_canopy(&micro);
      }
 if ( canopy[ (ix_stop-1 + ix_start)/2 + 1][ (iy_stop-1 + iy_start)/2 +2] == 2 )
  {
```

```
            canopy_triangle(&micro,
              (max_x + min_x) / 2, max_y,
              (max_x + min_x) / 2, (max_y + min_y) / 2,
              ((max_x + min_x) / 4) + (max_x / 2,((max_y + min_y) / 4) + (max_y / 2));
            ctdb_buffer_canopy(&micro);
          }
        if ( canopy[ ((ix_stop-1 + ix_start) / 2) - 1][ (iy_stop-1 + iy_start)/2 +2] == 2 )
          {
            canopy_triangle(&micro,
              (max_x + min_x) / 2, (max_y + min_y) / 2,
              ((max_x + min_x) / 4) + (min_x / 2, ((max_y + min_y) / 4) + (max_y / 2),
              (max_x + min_x) / 2, max_y);
            ctdb_buffer_canopy(&micro);
          }
        }
        else if ( canopy[ix_start][iy_start] == 0  &&canopy[ix_stop-1][iy_stop-1] == 2 )
        {
        reference = 0;
        edge.fullness = DEFAULT_FULLNESS;
        edge.start[X] = min_x; edge.start[Y] = min_y;
        edge.end[X] = max_x; edge.end[Y] = max_y;
        ctdb_buffer_cedge(&edge);
        canopy_triangle(&micro, max_x, min_y, min_x, max_y, max_x, max_y);
        ctdb_buffer_canopy(&micro);
        if ( canopy[ix_start][ (iy_stop-1+ iy_start)/2 + 1] == 2 )
          {
            canopy_triangle(&micro, min_x, (max_y + min_y) / 2 ,
              (max_x + min_x) / 2, (max_y + min_y) / 2,
              min_x, max_y);
            ctdb_buffer_canopy(&micro);
          }
        if ( canopy[ix_stop-1][iy_start+2] == 2 )
          {
            canopy_triangle(&micro,  (max_x + min_x) / 2,min_y;
              (max_x + min_x) / 2,(max_y + min_y) / 2
              max_x, min_y);
            ctdb_buffer_canopy(&micro);
          }
        }
    else if ( canopy[ix_stop-1][iy_start] == 0  &&canopy[ix_start][iy_stop-1] == 2 )
    {
    reference = 0;
    edge.fullness = DEFAULT_FULLNESS;
    edge.start[X] = min_x; edge.start[Y] = min_y;
    edge.end[X] = min_x; edge.end[Y] = max_y;
    ctdb_buffer_cedge(&edge);
    canopy_triangle(&micro, min_x, min_y,
    min_x, max_y, max_xmax_y);
    ctdb_buffer_canopy(&micro);

    if ( canopy[ (ix_stop-1 + ix_start)/2 + 1][ (iy_stop-1 + iy_start) / 2] == 2 )
      {
       canopy_triangle(&micro,  (max_x + min_x) / 2, (max_y + min_y) / 2,
          max_x, (max_y + min_y) / 2, max_x, max_y);
       ctdb_buffer_canopy(&micro);
      }
```

```
}
else if ( canopy[ix_stop-1][iy_stop-1] == 0 &&  canopy[ix_start][iy_start] == 2 )
  {
   reference = 0;
   edge.fullness = DEFAULT_FULLNESS;
   edge.start[X] = min_x; edge.start[Y] = min_y;
   edge.end[X] = min_x; edge.end[Y] = max_y;
   ctdb_buffer_cedge(&edge);
   canopy_triangle(&micro, min_x, min_y, min_x, max_y, max_x, min_y);
   ctdb_buffer_canopy(&micro);
   if ( canopy[ix_start+1][iy_stop-1] == 2 )
     {
        canopy_triangle(&micro,  min_x, max_y, (max_x + min_x) / 2, max_y,
          (max_x + min_x) / 2, (max_y + min_y) / 2);
        ctdb_buffer_canopy(&micro);
     }
   if ( canopy[ix_stop-1][iy_start + 2] == 2 )
     {
       canopy_triangle(&micro, (max_x + min_x) / 2,
        (max_y + min_y) / 2, max_x, (max_y + min_y) / 2
         max_x, min_y);
       ctdb_buffer_canopy(&micro);
     }
  }
else if ( canopy[ix_start][iy_stop - 1] == 0 )
  {
   edge.start[X] = max_x; edge.start[Y] = min_y;
   edge.end[X] = max_x; edge.end[Y] = max_y;
   ctdb_buffer_cedge(&edge);
   canopy_triangle(&micro, min_x, min_y,
      max_x, min_y, max_x, max_y);
   ctdb_buffer_canopy(&micro);
  }
else
  {
   edge.start[X] = max_x; edge.start[Y] = min_y;
   edge.end[X] = max_x; edge.end[Y] = max_y;
   ctdb_buffer_cedge(&edge);
   canopy_triangle(&micro, min_x, min_y, min_x, max_y, max_x,max_y);
   ctdb_buffer_canopy(&micro);
  }
}

}
```

A.16 Fit_vrt

```
static void fit_vrt()
  {
   int khills,ix,iy,iix,iiy,icx,icy,number_grid_squares;
   float64 my_resi[500][500],x_center,y_center;
   float64 max_z,dx,dy,dx1,dy1,deg,my_z,loc_x,loc_y;
   float64 current_max,current_min;
   int min_x,min_y,max_x,max_y;
   float64 len,len2,abs_z,rwx,rwy;
   int isplit,iterations;
```

```
int npx,npy,xstart,xstop,ystart,ystop;
int k,number_of_fits;
max_z = -999999.0;
npx = ctdb_in->max_x_post;
npy = ctdb_in->max_y_post;
for (iy = 0; iy < npy; iy++)
 {
  for (ix = 0; ix < npx; ix++)
   {
     my_resi[ix][iy] = original[ix][iy] - ctdb_in->min_z;
     temp[ix][iy] = 0.0;
     abs_z = sqrt(my_resi[ix][iy]*my_resi[ix][iy]);
     if ( abs_z > max_z)
      {
        max_z = abs_z;
        icx = ix;
        icy = iy;
      }
   }
 }
khills = 0;
isplit = 2;
while (isplit < 32)
{
if (isplit < 8) number_of_fits = 4;
if (isplit >= 8) number_of_fits =2;
number_grid_squares = npx/isplit;
x_center = (float)number_grid_squares*ctdb_in->incrD/2.0;
for (iiy = 0; iiy < isplit; iiy++)
 {
   ystart = iiy*number_grid_squares;
   ystop = (iiy + 1)*number_grid_squares;
   if (ystop > npy) ystop = npy;
   for (iix = 0; iix < isplit; iix++)
    {
      xstart = iix*number_grid_squares;
      xstop = (iix + 1)*number_grid_squares;
      if (xstop > npx) xstop = npx;
      for (k = 0; k < number_of_fits;k++)
       {
         loc_y = (float)iiy*2.0*x_center + x_center;
         loc_x = (float)iix*2.0*x_center + x_center;
         hill[khills].hgt = 0.0;
         hill[khills].wgty = 0.4*drand48() + 0.1;
         current_min = 9999999.0;
         current_max = -9999999.0;
         for (iy = ystart; iy < ystop;iy++)
          {
           for (ix = xstart; ix < xstop;ix++)
            {
             if (my_resi[ix][iy] < current_min)
              {
                current_min = my_resi[ix][iy];
                min_x = ix;
                min_y = iy;
              }
```

```
          if (my_resi[ix][iy] > current_max)
           {
             current_max = my_resi[ix][iy];
             max_x = ix;
             max_y = iy;
           }
         }
        }
      if ( current_max*current_max > current_min*current_min)
       {
         hill[khills].hgt = my_resi[max_x][max_y];
         hill[khills].cx = (float)max_x*ctdb_in->incrD;
         hill[khills].cy = (float)max_y*ctdb_in->incrD;
         my_z = current_min;
       }
      else
       {
         hill[khills].hgt = my_resi[min_x][min_y];
         hill[khills].cx = (float)min_x*ctdb_in->incrD;
         hill[khills].cy = (float)min_y*ctdb_in->incrD;
         my_z = current_max;
       }
      hill[khills].power = (0.3 + 0.5*drand48());
      len = sqrt((float)((max_x - min_x)*(max_x - min_x) +
      (max_y - min_y)*(max_y - min_y)));
      len = len*ctdb_in->incrD;
      hill[khills].wgtx =
         ((xstop - xstart)*ctdb_in->incrD)*(0.7 + 0.3*drand48());
      dx = (float) (max_x - min_x);
      dy = (float) (max_y - min_y);
      if (dx*dx > 0.001)
       {
         hill[khills].angle = 90.0 - atan(dy/dx)*45.0/atan(1.0);
       }
      else
       {
         hill[khills].angle = 0.0;
       }
      hill[khills].hgt = hill[khills].hgt + ctdb_in->min_z;
      for (iy = 0; iy < npy; iy++)
       {
         loc_y = iy*ctdb_in->incrD;
         for (ix = 0; ix < npx; ix++)
           {
             loc_x = ix*ctdb_in->incrD;
             temp[ix][iy] = temp[ix][iy] + single_hill(loc_x,loc_y,khills);
             my_z = temp[ix][iy] + ctdb_in->min_z;
             my_resi[ix][iy] = original[ix][iy] - my_z;
           }
       }
      khills = khills +1;
       }
      }
    }
  isplit = isplit*2;
}
```

```
for (nhills = khills;nhills < 1000;nhills++)
 {
   loc_x = icx*ctdb_in->incrD;
   loc_y = icy*ctdb_in->incrD;
   hill[nhills].cx = loc_x;
   hill[nhills].cy = loc_y;
   hill[nhills].hgt = my_resi[icx][icy] + ctdb_in->min_z;
   printf("hill # %d cx %d cy %d %.3lf \n",nhills,icx,icy,my_resi[icx][icy]);
   hill[nhills].angle = 180*drand48();
   hill[nhills].power = 0.7*drand48() + 0.1;
   hill[nhills].wgtx = 0.10*drand48()*(ctdb_in->max_x - ctdb_in->min_x)
   + 0.005*(ctdb_in->max_x - ctdb_in->min_x);
   hill[nhills].wgty = 0.8*drand48() + 0.05;
   max_z = -999999.0;
   for (iy = 0; iy < npy; iy++)
    {
      loc_y = iy*ctdb_in->incrD;
      for (ix = 0; ix < npx; ix++)
       {
         loc_x = ix*ctdb_in->incrD;
         temp[ix][iy] = temp[ix][iy] + single_hill(loc_x,loc_y,nhills);
         my_z = temp[ix][iy] + ctdb_in->min_z;
         my_resi[ix][iy] = original[ix][iy] - my_z;
         abs_z = sqrt(my_resi[ix][iy]*my_resi[ix][iy]);
         if ( abs_z > max_z)
          {
            max_z = abs_z;
            icx = ix;
            icy = iy;
          }
         }
    }
 }
}
```

A.17 VRT

static double vrt(px,py,khills)

```
    int khills;
    double px,py;
   {
    double my_z,current_term;
    int ih;
    my_z = 0.0;
    current_term = 0.0;
    for (ih = 0; ih <= khills; ih++)
       {
        current_term = single_hill(px,py,ih);
        my_z = my_z + current_term;
       }
    my_z = my_z + ctdb_in->min_z;
    return my_z;
   }
```

A.18 Single_hill

static double single_hill(px,py,ih)

```
    int ih;
```

```
double px,py;
{
```

/* The function single_hill evaluates the expression:

$$h(px, py) = hgt e^{-wgtx[(X(px,py)-X(cx,cy))^2 + wgty(Y(px,py)-Y(cx,cy))^2]^{power}}$$, where

$$X(x,y) = \cos(\omega)x + \sin(\omega)y \text{ and } Y(x,y) = -\sin(\omega)x + \cos(\omega)y$$ */

```
 double my_x,my_y,my_single_hill_z;
 double xxx,yyy,xx,yy,maxh,minh,scale_x,scale_y;
 double cx,cy,cz,c,s,xx0,yy0,dxx,dyy,expon,zz;
 double aa,bb,my_w,my_sign,computed_wgt;
 double current_term,offset_x,offset_y,hgt,hgtm;
 double nd_length_of_axis;
 double deg_to_rad = atan(1.0)*4.0/180.0;
 double my_max_z,my_min_z;
 int32 my_max_x, my_max_y, my_min_x, my_min_y;
 my_x = (px - ctdb_in->min_x)/(ctdb_in->max_x - ctdb_in->min_x);
 my_y = (py - ctdb_in->min_y)/(ctdb_in->max_y - ctdb_in->min_y);
 c = cos(deg_to_rad*hill[ih].angle);
 s = sin(deg_to_rad*hill[ih].angle);
 cx = (hill[ih].cx - ctdb_in->min_x)/(ctdb_in->max_x - ctdb_in->min_x);
 cy = (hill[ih].cy - ctdb_in->min_y)/(ctdb_in->max_y - ctdb_in->min_y);
 cz = (hill[ih].hgt - ctdb_in->min_z)/hgt_max;
 nd_length_of_axis =
      (hill[ih].wgtx - ctdb_in->min_x)/(ctdb_in->max_x - ctdb_in->min_x);
 yy0 = cx*c + cy*s;
 xx0 = -cx*s + cy*c;
 my_sign = 1.0;
 if ( hill[ih].hgt < 0.0) my_sign = -1.0;
 aa = log(0.001/sqrt(cz*cz));
 bb = log(nd_length_of_axis*nd_length_of_axis)*hill[ih].power;
      computed_wgt = -1.0*aa/exp(bb);
 yy = my_x*c + my_y*s;
 xx = -my_x*s + my_y*c;
 dxx = (xx - xx0)*(xx - xx0);
 dyy = (yy - yy0)*(yy - yy0);
 expon = exp(hill[ih].power*log(dxx + hill[ih].wgty*dyy));
 current_term = sqrt(cz*cz)*exp(-1.0*computed_wgt*expon);
 current_term = current_term*hgt_max ;
 my_single_hill_z = my_sign*current_term;
 return my_single_hill_z;
}
```

A.19 Add_Subcanopies

```
static void add_subcanopies (patch_water_state)
    int32 *patch_water_state;
    {
     CTDB_SEARCH_SPACE_PTR_CMP search_space;
     int32 i, j,ii,jj;
     int32 count;
     int32 my_type;
     CTDB_ABSTRACT_DATA_CMP *data;
     int32 data_size;
```

```
float32 *my_vert= NULL;
int32 vertex_number = 0;
 float64 meters_per_patch;
meters_per_patch = ctdb_out.incrD * ctdb_out.patch_incrD;
for ( ii = 0;ii < soil_region;ii++)
   {
     vertex_number = my_reg[ii].count;
     for ( jj = 0; jj < vertex_number; jj++)
       {
          my_vert[2*jj] = my_reg[ii].x[jj];
          my_vert[2*jj + 1] = my_reg[ii].y[jj];
       }
     my_type = CTDB_ABSTRACT_SOIL_DEFRAG_CMP;
     data_size = 1;
     data->soil_defrag.soil_index = 9;
     data->soil_defrag.level = 0;
     data_size = 1;
     ctdb_store_abstract(&ctdb_out, &quad_root, my_type,
       vertex_number, my_vert, data_size, (int32 *)data,
       min_patch_x * meters_per_patch,
       min_patch_y * meters_per_patch,
       max_patch_x * meters_per_patch,
       max_patch_y * meters_per_patch);
   }

 if (num_lverts > 0) free(lverts);
}
```

APPENDIX B

A SAMPLE ENTITY PARAMETER FILE

INTENTIONALLY LEFT BLANK

A SAMPLE ENTITY PARAMETER FILE

This appendix contains the entity parameter file for an experimental UGV. The file resides in the directory "/modsaf3.0/common/src/ModSAF/entities," where "/modsaf3.0/" is the root directory for the ModSAF simulation code. In the following text, a double semi-colon (;;) indicates a comment in the file. Some formatting and bold print have been added to the file for clarity. The reader is referred to the ModSAF references (Sagacitech 1997, and Smith 1995) for more information about entity parameter files.

```
;;US_UGV_M_T_A_params.rdr

US_UGV_M_T_A_MODEL_PARAMETERS {
;; 1. ENTITY PARAMENTERS
(SM_Entity              DEFAULT_DEAD_RECKONING_PARAMETERS
                        (vehicle_class vehicleClassSimple)
                        (guises vehicle_US_UGV vehicle_US_HMMWV)
                        (send_dis_deactivate true)  )

;; 2. VULNERABILITY MODELS & DAMAGE ASSESMENT
(SM_DFDamage            (filename "dfdam_TRUCK.rdr")
                         (damage_threshold 10.0))
(SM_IFDamage             (name apc1))
 (SM_VAssess            (background on)
                        (sensors commander-sight)
                         (weapons )
                        VASSESS_ADA_GROUND
                        VASSESS_ADA_THREATS
                        VASSESS_IFV_OPTIONAL
                        (no_target_load)  )

;; 3. VEHICLE COMPONENTS
(SM_Components          (hull SM_TrackedHull SAFCapabilityMobility)
                        (primary-turret [SM_GenericTurret | 0] )
                        (commander-sight [SM_Visual | 1] )
                        (gunner-sight [SM_Visual | 0] )
                        (radioA [SM_GenRadio | 0] )
                        (radioB [SM_GenRadio | 2] ) )

(SM_EnvAssess            (commander_sight "commander-sight"))
(SM_EnvReason )

;;4. PATH PLANNING  PARAMETERS
(SM_LocalMap            (skirt_deviation 0.3))
 (SM_VMove              (background on)
                        (stopped_time 60.0)
                        (default_speed 10.0)
                        (default_max_deviation 1000.0)
                        (default_catchup_speed 0.0)
                        (default_brake_strength 1.0)
                        (max_backup_distance 1.0)
                        (planning_horizon 60.0)
                        (execution_horizon 1.0)
```

```
                    (moving_obstacle_horizon 10.0)
                    (env_sampling_period 6) )
```

;; 5. TERRAIN ANALYSIS PARAMETERS

```
(SM_Vterrain        (entity_period 100)
                    (avoidance_mask [VTERRAIN_BUILDING |
                                     VTERRAIN_WATER |
                                     VTERRAIN_DITCH] )
                    (avoid_soils SOIL_DEEP_WATER 123)
                    (background on)
                    (movement_threshold 1.0)
                    (map_radius 500.0)
                    (entity_radius 10.0)
                    (history_list_spacing 20.0)
                    (num_history_list_points 50)
                    (breach_obst_nominal_size 400.0))
```

;; 6. MOBILITY PARAMETERS

```
(SM_TrackedHull     (mobility_model 1)
        (soils
           (SOIL_DEFAULT  (max_speeds 52.0 52.0)  SOIL_DEFAULT_TRACKED)

              ( 1   (max_speeds 52.0 52.0)   (max_accel 1.32)
                    (max_decel 5.37)         (max_turn 47.0)
                    (max_climb 36.0)         (dust_speeds 100.0 100.0 100.0) )

              ( 2   (max_speeds 52.0 52.0)   (max_accel 1.32)
                    (max_decel 5.37)         (max_turn 47.0)
                    (max_climb 36.0)         (dust_speeds 100.0 100.0 100.0) )

              ( 3   (max_speeds 52.0 52.0)   (max_accel 1.32)
                    (max_decel 5.37)         (max_turn 47.0)
                    (max_climb 36.0)         (dust_speeds 100.0 100.0 100.0) )

              ( 4   (max_speeds 52.0 52.0)   (max_accel 1.32)
                    (max_decel 5.37)         (max_turn 47.0)
                    (max_climb 36.0)         (dust_speeds 100.0 100.0 100.0) )

              ( 5   (max_speeds 52.0 52.0)   (max_accel 1.32)
                    (max_decel 5.37)         (max_turn 47.0)
                    (max_climb 36.0)         (dust_speeds 100.0 100.0 100.0) )

              ( 6   (max_speeds 52.0 52.0)   (max_accel 1.32)
                    (max_decel 5.37)         (max_turn 47.0)
                    (max_climb 36.0)         (dust_speeds 100.0 100.0 100.0) )

              ( 7   (max_speeds 52.0 52.0)   (max_accel 1.32)
                    (max_decel 5.37)         (max_turn 47.0)
                    (max_climb 36.0)         (dust_speeds 100.0 100.0 100.0) )

              ( 8   (max_speeds 52.0 52.0)   (max_accel 1.32)
                    (max_decel 5.37)         (max_turn 47.0)
                    (max_climb 36.0)         (dust_speeds 100.0 100.0 100.0) )

              ( 9   (max_speeds 8.05 8.05)   (max_accel 1.32)
                    (max_decel 5.37)         (max_turn 47.0)
```

```
               (max_climb 36.0)          (dust_speeds 100.0 100.0 100.0) )

         ( 10  (max_speeds 52.0 52.0)    (max_accel 1.32)
               (max_decel 5.37)          (max_turn 47.0)
               (max_climb 36.0)          (dust_speeds 100.0 100.0 100.0) )

         ( 11  (max_speeds 52.0 52.0)    (max_accel 1.32)
               (max_decel 5.37)          (max_turn 47.0)
               (max_climb 36.0)          (dust_speeds 100.0 100.0 100.0) )

         ( 12  (max_speeds 52.0 52.0)    (max_accel 1.32)
               (max_decel 5.37)          (max_turn 47.0)
               (max_climb 36.0)          (dust_speeds 100.0 100.0 100.0) )

         ( 13  (max_speeds 52.0 52.0)    (max_accel 1.32)
               (max_decel 5.37)          (max_turn 47.0)
               (max_climb 36.0)          (dust_speeds 100.0 100.0 100.0) )

         ( 14  (max_speeds 52.0 52.0)    (max_accel 1.32)
               (max_decel 5.37)          (max_turn 47.0)
               (max_climb 36.0)          (dust_speeds 100.0 100.0 100.0) )

         ( 15  (max_speeds 52.0 52.0)    (max_accel 1.32)
               (max_decel 5.37)          (max_turn 47.0)
               (max_climb 36.0)          (dust_speeds 100.0 100.0 100.0) )

         ( 16  (max_speeds 52.0 52.0)    (max_accel 1.32)
               (max_decel 5.37)          (max_turn 47.0)
               (max_climb 36.0)          (dust_speeds 100.0 100.0 100.0) ) )

              (fuel_usage               (0.0 100.0) (0.125 12.0)))
```

;; 7. TURRET PARAMETERS

```
([SM_GenericTurret | 0]   (physdb_name "primary-turret")
                           (rates continuous 0.0 40.0))
```

;; 8. VISUAL SYSTEMS

```
([SM_Visual | 0]          VISUAL_APC_DRIVER_DVO_NVO )
([SM_Visual | 1]          VISUAL_LOSAT_HIRES_IR )

(SM_SubComp)

(SM_VSpotter              (background on)
                          (sensors commander-sight)
                          VSPOTTER_SPECS )
```

;; 9. RADIOS

```
([SM_GenRadio | 0]        (net_name  "platoon_net")
                          (aspid ASPID_MODSAF_TEXT)
                          GENRADIO_BLUE_PARAMS  )
([SM_GenRadio | 2]        (net_name  "company_net")
                          (aspid ASPID_MODSAF_TEXT)
                          GENRADIO_BLUE_PARAMS )
```

;; 10. VISUAL SEARCH ALGORITHMS

```
(SM_Vsearch      (search_type ground)
                 (scan_mode static)
                 (background on)
```

```
        (turret_scanner "primary-turret" 9.0 5.0)
        (gun_scanner    "")
        (visual_scanners "commander-sight")
        (stopped_duty_cycle   1.0)
        (moving_duty_cycle    0.0)
        (restrict2for         "none"))

}
```

NO. OF COPIES	ORGANIZATION
2	ADMINISTRATOR DEFENSE TECHNICAL INFO CENTER ATTN DTIC OCP 8725 JOHN J KINGMAN RD STE 0944 FT BELVOIR VA 22060-6218
1	DIRECTOR US ARMY RESEARCH LABORATORY ATTN AMSRL CS AL TA REC MGMT 2800 POWDER MILL RD ADELPHI MD 20783-1197
1	DIRECTOR US ARMY RESEARCH LABORATORY ATTN AMSRL CI LL TECH LIB 2800 POWDER MILL RD ADELPHI MD 207830-1197
1	DIRECTOR US ARMY RESEARCH LABORATORY ATTN AMSRL DD J J ROCCHIO 2800 POWDER MILL RD ADELPHI MD 20783-1197
1	OSD OUSD(A&T)/ODDDR&E(R) ATTN R J TREW THE PENTAGON WASHINGTON DC 20301-7100
1	US ARMY INFO SYS ENGRG CMND ATTN ASQB OTD F JENIA FT HUACHUCA AZ 85613-5300
1	AMCOM MRDEC ATTN AMSMI RD W C MCCORKLE REDSTONE ARSENAL AL 35898-5240
1	US ARMY TANK-AUTOMOTIVE CMD RD&E CTR ATTN AMSTA TA J CHAPIN WARREN MI 48397-5000
1	INST FOR ADVNCD TCHNLGY THE UNIV OF TEXAS AT AUSTIN PO BOX 20797 AUSTIN TX 78720-1714
1	CECOM ATTN PM GPS COL S YOUNG FT MONMOUTH NJ 07703

NO. OF COPIES	ORGANIZATION
1	DARPA ATTN B KASPAR 3701 N FAIRFAX DR ARLINGTON VA 22203-1714
1	CECOM SP & TERRESTRIAL COMMCTN DIV ATTN AMSEL RD ST MC M H SOICHER FT MONMOUTH NJ 07703
1	NAVAL SURFACE WARFARE CTR CODE B07 J PENNELLA 17320 DAHLGREN RD BLDG 1470 RM 1101 DALGREN VA 224480-5100
1	US ARMY NATICK RDEC ACTING TECHNICAL DIR ATTN SSCNC T P BRANDLER NATICK MA 01760-5002
1	US MILITARY ACADEMY MATH SCI CTR OF EXCELLENCE DEPT OF MATHEMAICAL SCI MAJ MD PHILLIPS THAYER HALL WEST POINT NY 10996-1786
1	US ARMY TRAINING & DOCTRINE CMD BATTLE INTEGRATION & TECH DIR ATTN ACTD B J A KLEVECZ FT. MONROE VA 23651-5850
2	COMMANDER US ARMY TACOM J JACZKOWSKI WARREN MI 48397-5000
1	UNIV OF TEXAS ARL ELECTROMAG GROUP CAMPUS MAIL CODE F0250 ATTN A TUCKER AUSTIN TX 78713-8029
2	DIRECTOR US ARMY WES R AJLVIN 3909 HALLS FERRY ROAD VICKSBURG MS 39180-6199
1	HICKS & ASSOCIATES, INC. ATTN G SINGLEY III 1710 GOODRICH DR STE 1300 MCLEAN VA 22102

NO. OF COPIES	ORGANIZATION
2	NIST ATTN K MURPHY 100 BUREAU DRIVE GAITHERSBURG MD 20899
1	SPECIAL ASST TO THE WING CDR 50SW/CCX CAPT P H BERSTEIN 300 O'MALLEY AVE STE 20 FALCON AFB CO 80912-3020
2	US ARMY MOUNTED MANEUVER BATTLELAB ATTN MAJ J BURNS BLDG 2021 BLACKHORSE REGIMENT DR FORT KNOX KY 40121
1	HQ AFWA/DNX 106 PEACEKEEPER DR STE 2N3 OFFUTT AFB 68113-4039
2	NASA JET PROPULSION LAB ATTN L MATHIES K OWENS 4800 OAK GROVE DR PASADENA CA 91109
1	US ARMY RESEARCH OFC 4300 S MIAMI BLVD RESEARCH TRIANGLE PARK NC 27709
1	US ARMY SIMULATION TRAIN & INSTRMNTN CMD ATTN J STAHL 12350 RESEARCH PARKWAY ORLANDO FL 32826-3726
1	US ARMY TANK-AUTOMOTIVE & ARMAMENTS CMD ATTN AMSTA AR TD M FISETTE BLDG 1 PICATINNY ARSENAL NJ 07806-5000
1	DPTY CG FOR RDE US ARMY MATERIEL CMD ATTN AMCRD MG CALDWELL 5001 EISENHOWER AVE ALEXANDRIA VA 22333-0001
4	ROBOTICS SYSTEMS TECNOLOGY INC ATTN S MYERS P CORY B BEESON J ROBERTSON 1234 TECH COURT WESTMINISTER MD 21157

NO. OF COPIES	ORGANIZATION
	ABERDEEN PROVING GROUND
2	DIRECTOR US ARMY RESEARCH LABORATORY ATTN AMSRL CI LP (TECH LIB) BLDG 305 APG AA
4	US ARMY EDGEWOOD RDEC ATTN SCBRD TD J VERVIER APG MD 21010-5423
4	CDR USA ATC ATTN STEAC CO COL ELLIS STEAC TD J FASIG STEAC TE R CUNNINGHAM STEAC RM A MOORE BLDG 400
2	CDR USA ATC ATTN STEAC TE F P OXENBERG STEAC TE F A SCRAMLIN BLDG 321
3	CDR USA TECOM ATTN AMSTE CD B SIMMONS AMSTE CD R COZBY J HAUG RYAN BLDG
1	DIR USARL ATTN AMSRL WM I MAY BLDG 4600
2	DIR USARL ATTN AMSRL WM B A W HORST JR W CIEPIELLA BLDG 4600
1	DIR USARL ATTN AMSRL WM BA W D'AMICO BLDG 4600
1	DIR USARL ATTN AMSRL WM BC P PLOSTINS BLDG 390
1	DIR USARL ATTN AMSRL WM BD B FORCH BLDG 4600
1	DIR USARL AMSRL WM BE G WREN BLDG 390

NO. OF COPIES	ORGANIZATION
18	DIR USARL ATTN AMSRL WM BF J LACETERA R PEARSON P CORCORAN M FIELDS (15 CYS) BLDG 120
3	DIR USARL ATTN AMSRL WM BB H ROGERS B HAUG J BORNSTEIN BLDG 1121
1	DIR USARL ATTN AMSRL WM BB G HAAS BLDG 1120A
1	DIR USARL ATTN AMSRL WM BR C SHOEMAKER BLDG 1121
	ABSTRACT ONLY
1	DIRECTOR US ARMY RESEARCH LABORATORY ATTN AMSRL CS AL TP TECH PUB BR 2800 POWDER MILL RD ADELPHI MD 20783-1197

INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	August 1999	Final

4. TITLE AND SUBTITLE
Modifying ModSAF Terrain Databases to Support the Evaluation of Small Weapons Platforms in Tactical Scenarios

5. FUNDING NUMBERS
PR: 1L162618AH80

6. AUTHOR(S)
Fields, M.A. (ARL)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
Weapons and Materials Research Directorate
Aberdeen Proving Ground, MD 21010-5066

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
Weapons and Materials Research Directorate
Aberdeen Proving Ground, MD 21010-5066

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
ARL-TR-1996

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

In this report, we describe tools for the creation and modification of modular semi-autonomous forces (ModSAF) terrain databases to support the evaluation of a small autonomous robot in a tactical scenario. Our work is motivated by the modeling and simulation needs of the Demo III robotics program which is developing a small tactical robot called the experimental unmanned vehicle (XUV). The XUV is a small wheeled robot which must autonomously navigate through its environment. The primary mission of the XUV will be to augment the scout forces, so it must provide reconnaissance, surveillance, and target acquisition information (RSTA) to its operators. Modeling the XUV in a simulated environment is challenging since existing terrain databases do not have sufficient resolution to examine the mobility characteristics of small vehicles.

Our tools increase the resolution and detail of existing terrain databases so that the databases have sufficient detail to challenge the mobility, chassis dynamics, and RSTA models of a small unmanned platform. To properly model a small vehicle such as the XUV, the terrain database in ModSAF needs to be modified. The modification is done in two phases. In the first phase, the resolution of the grid underlying the terrain is increased by placing additional elevation grid posts between the existing posts. Elevations are assigned to the new grid posts using mathematical terrain models such as the variable resolution terrain Model (Wald & Patterson, 1992). The new, higher resolution terrain directly affects the vehicle dynamics and the line-of-sight algorithms. The new terrain does not directly affect the ModSAF route-planning algorithms. In the second phase of our terrain database modifications, the slopes on the new terrain are examined. Regions that are steep or inaccessible to the XUV are marked as obstacles in the database. The route-planning algorithms use these "obstacles" to avoid planning routes through regions that are too steep for the XUV.

14. SUBJECT TERMS
force modeling terrain database
simulation

15. NUMBER OF PAGES
60

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102